RF Music Festival: orquesta basada en microcontroladores PIC18 y RF

Juan Gallostra Acín

Febrero 2015

Abstract

El presente trabajo de final de grado desarrolla la recreación, mediante el uso de microcontroladores comunicados por radiofrecuencia, de una orquesta compuesta por un director y cuatro músicos capaz de reproducir piezas musicales.

Se detallan y explican, de cada uno de los integrantes de la orquesta, sus componentes y las conexiones entre estos así como el código escrito y utilizado con los objetivos de establecer una red de comunicación maestro-esclavos para la transmisión de información y la generación del sonido acorde al instrumento y nota de cada músico a lo largo de la ejecución.

Además se valida también el resultado mediante la interpretación de la orquesta de las piezas *Bourrée I* de la Suite No. 2 en Si menor y *Air on the G string*, ambas de J.S.Bach, y se analizan los costes e impacto medioambiental del proyecto aquí presentado.

Índice

Índice de Figuras

Índice de Tablas

1 Glosario

- ADC : *Analog-to-Digital Conversion*
- BSD : *Berkeley Software Distribution*
- CE : *Chip Enable*
- CONFIG : *Configuration*
- CPU : *Central Processing Unit*
- CRC : *Cyclic Redundancy Check*
- DAC : *Digital-to-Analog Conversion*
- EN_ADDR : *Enable Address*
- FIFO : *First In First Out*
- GFSK : *Gaussian Frequency Shift Keying*
- GND : *Ground*
- GPIO : *General Purpose Input Output*
- I2C : *Inter-Integrated Circuit*
- IRQ : *Interrupt Request*
- ISM : *Industrial, Scientific and Medical*
- KiB : *Kibibyte*
- LNA : *Low Noise Amplifier*
- LSByte : *Least Significant Byte*
- Mbps : *Megabits per second*
- MISO : *Master In Slave Out*
- MOSI : *Master Out Slave In*
- MSB : *Most Significant Bit*

- NSS : *Not Slave Select*
- PRIM_RX : *Primary Receiver*
- PWM : *Pulse Width Modulation*
- PWR_UP : *Power Up*
- RAEE : Residuos de aparatos eléctricos y electrónicos
- RAM : *Random Access Memory*
- RC : *Resistance-Capacitor*
- RF : Radio Frecuencia
- ROM : *Read Only Memory*
- RTOS : *Real Time Operating System*
- RX : Receptor
- SAR : *Specific Absortion Rate*
- SDI : *Slave Data In*
- SDO : *Slave Data Out*
- SCK : *Serial Clock*
- SPI : *Serial Peripherial Interface*
- SSPBUF : *Serial Input Buffer*
- SSPSR : *Shift Register*
- TX : Transmisor
- UART : *Universal Asynchronous Receiver-Transmitter*
- USB : *Universal Serial Bus*

2 Introducción

El proyecto nace a partir de una propuesta del profesor Manuel Moreno-Eguilaz, del Departamento de Ingeniería Electrónica de la ETSEIB UPC, en la bolsa de proyectos de dicha escuela y del interés del estudiante del Grado en Tecnologías Industriales Juan Gallostra por desarrollarla.

Este proyecto pretende recrear una orquesta basada en microcontroladores comunicados mediante Radio Frecuencia (RF) capaz de interpretar piezas musicales.

La orquesta realizada en este proyecto está formada por cinco integrantes: un director y cuatro músicos. Los cuatro músicos se diferencian únicamente en el instrumento asignado a cada uno, ya que todos ellos comparten una misma función, mientras que el director se diferencia de los músicos no sólo en su función si no que también en sus componentes. Los instrumentos presentes en la orquesta son: un bajo, un violín y dos guitarras.

El director de la orquesta es el que se encarga de organizar y gestionar el funcionamiento de la orquesta. Él es el encargado de comprobar que todos los músicos están activos y de transmitir a cada uno de ellos sus notas e instrucciones relacionadas con la ejecución de la pieza en particular.

El microcontrolador utilizado como cerebro de cada músico ha sido el PIC18F4520, mientras que para el director se ha utilizado un microcontrolador de 32 bits incluido en la placa Pyboard. Para la comunicación entre ellos se ha utilizado el módulo de RF nRF24L01+. Por último, para la obtención del sonido en formato analógico de cada nota se ha partido de una señal digital generada por la salida PWM a la que se ha conectado un pequeño filtro RC pasabajos que a su vez se ha conectado a unos altavoces que incluyen un pequeño amplificador.

El código que se ha desarrollado y adaptado permite realizar las tres tareas fundamentales para el funcionamiento de la orquesta: codificar las partituras e instrucciones de cada músico y preparar los mensajes que se enviarán por RF, establecer una vía de comunicación vía RF y descodificar de los mensajes y actuar de acuerdo a su contenido.

2.1 Objetivos del proyecto

El objetivo principal del proyecto, como se ha explicado previamente, consiste en recrear electrónicamente una orquesta capaz de interpretar piezas musicales con la peculiaridad de que la forman únicamente microcontroladores. Este objetivo, sin embargo, se puede concretar en los siguientes:

- Diseñar los músicos de la orquesta eligiendo o fabricando los componentes necesarios para que realicen las funciones que se les han asignado: enviar y recibir información por RF y generar los sonidos de las notas pertenecientes a su partitura.
- Diseñar el director de la orquesta eligiendo o fabricando los componentes necesarios para que realice también las funciones que se le han asignado: enviar y recibir información por RF y ser capaz de almacenar, para una pieza musical, las partituras de todos los músicos.
- Establecer una red de transmisión de información vía RF entre los microcontroladores que conforman la orquesta.
- Preparar la partitura asociada a cada instrumento en un formato que permite su envío por RF y sea comprensible para el músico a que va dirigido.[1]
- Extraer la información recibida vía RF y o bien procesar la orden recibida o bien generar la señal PWM de acuerdo con el instrumento y nota del músico en cuestión.[2]

2.2 Alcance del proyecto

Se da el proyecto por finalizado de forma satisfactoria cuando la orquesta realizada y definida con anterioridad sea capaz de ejecutar, de forma sincronizada y sin errores, una pieza musical de longitud indeterminada en la que todos los músicos formen parte.

[1] Esta tarea sólo corresponde al director.
[2] Esta tarea sólo corresponde a los músicos.

3 Análisis de antecedentes

En la primera fase del proyecto se ha tratado de reunir toda la información necesaria para adquirir los conocimientos y recursos que facilitaran e hicieran posible el cumplimiento de los objetivos del proyecto. Este trabajo se ha separado principalmente en dos vertientes claramente diferenciadas, aunque ambas comprenden el estudio y la familiarización con las diferentes herramientas utilizadas para desarrollar el proyecto.

La primera vertiente engloba los componentes electrónicos, sus protocolos de comunicación asociados y los circuitos utilizados en el proyecto y la segunda tanto el código necesario para que los primeros cumplan con las tareas asignadas como los entornos de programación en los que este se de desarrolla.

De los componentes utilizados se ha profundizado, en el caso de los microcontroladores, en aquellas partes del manual que tratan temas relacionados con el proyecto tales como la comunicación SPI o la frecuencia de trabajo. En el caso del módulo nRF24L01+ se ha tratado con todo el manual y se han estudiado otros proyectos realizados con él. En cuanto al código, para las instrucciones recibidas por los músicos y el procedimiento utilizado para generar los sonidos de cada nota se ha aprovechado el trabajo realizado por Victor Timofeev. Para facilitar la gestión de los recursos del microcontrolador se ha hecho uso del sistema operativo para microcontroladores PIC OSA-RTOS. En cuanto al envío y recepción de paquetes de datos por RF a través del módulo nRF24L01+ se ha trabajado, por parte de los microcontroladores PIC18F4520 a partir de una librería desarrollada por Pablo Sanz en su PFC *Comunicación por RF entre microcontroladores PIC18 mediante el módulo NRF24L01* y facilitada por el profesor Manuel Moreno-Eguilaz y por parte de la Pyboard mediante un driver desarrollado por Damien George[1] de libre descarga a través de internet.

Puesto que los componentes y dispositivos electrónicos de cada miembro de la orquesta se explican en detalle en sus respectivos apartados se ha considerado aquí explicar brevemente aquellos sobre los que se ha investigado que, sin ser esenciales, sí que hacen que el proyecto sea posible.

[1] Damien George es también la persona que ha desarrollado y diseñado la Pyboard.

3.1 La comunicación vía SPI

La comunicación entre el microcontrolador y el módulo nRF24L01+ se lleva a cabo, principalmente, mediante un bus SPI. A través de este protocolo de comunicación se envían instrucciones, se configura y se puede conocer el estado del proceso llevado a cabo por el nRF24L01+. Garantizar su funcionamiento es imprescindible, pues su incorrecta configuración o uso ineficiente puede llevar al microcontrolador a malinterpretar el estado del nRF24L01+ y concluir que ha habido fallos en el envío y recepción de datos vía RF cuando no los ha habido.

Las características principales del protocolo SPI son las siguientes:

- El tipo de comunicación establecida entre los dos o más componentes que participan del intercambio de datos es siempre del tipo Maestro-Esclavo/s. El maestro es el encargado de iniciar la comunicación y generar la señal de reloj para sincronizar el envío y recepción de datos. En el caso de que la comunicación sea Maestro-Esclavos también es tarea suya habilitar o deshabilitar cada uno de los dispositivos esclavos.

- Es un protocolo de comunicación serie síncrono. Serie quiere decir que la información intercambiada se transmite mediante una línea donde los datos enviados o recibidos se multiplexan en el tiempo. Esto implica que la transmisión es bit a bit. Síncrono porque, paralelamente a la línea de transmisión de datos existe una línea de reloj que va del maestro al esclavo encargada de marcar los momentos de inicio y fin de transmisión de cada bit.

- Normalmente se establece mediante 4 líneas, aunque sólo 3 son estrictamente necesarias:

  1. **MOSI** (Master Out Slave In): Línea que sale del dispositivo maestro y entra en el esclavo. Es una de las líneas de transmisión de datos. A veces se le llama también SDO (Slave Data Out).
  2. **MISO** (Master In Slave Out): Línea que sale del dispositivo esclavo y entra en el maestro. Es la otra línea de transmisión de datos. A veces se le llama también SDI (Slave Data In).
  3. **NSS** (Not Slave Select): Línea encargada de activar o desactivar el dispositivo esclavo. Se le llama también $\overline{SS}$ y es una señal en lógica negativa.
  4. **SCK** (Serial Clock): Línea que sincroniza la transmisión de datos mediante una señal de reloj. La frecuencia de reloj debe ser la adecuada para permitir que el dispositivo esclavo

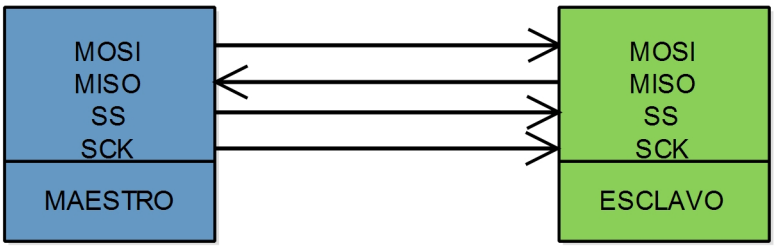

Figura 3.1: Conexiones entre dispositivos para la comunicación vía SPI.

- El estándar SPI trabaja en modo full dúplex, por lo que cuenta con dos vías de datos unidireccionales en vez de con una bidireccional. Una vía va del maestro al esclavo y la otra del esclavo al maestro. Estas dos líneas, además, no funcionan de forma independiente, si no que siempre se activan a la par. Esto implica que siempre que se envíe información por una línea se estará recibiendo a la vez por la otra, aunque esta no sea relevante. Cuando sólo interesa enviar o recibir se envían datos desechables por la otra línea que no serán procesados.
- Se trabaja con unidades de información de un byte. Cada dispositivo dispone de un registro llamado *Shift Register* que por un lado se vacía empezando por el MSB y se llena por el otro lado con este nuevo MSB. Esto implica que, antes de desactivar la señal NSS se hayan de completar un mínimo de 8 ciclos de reloj para que la transmisión del byte se complete.

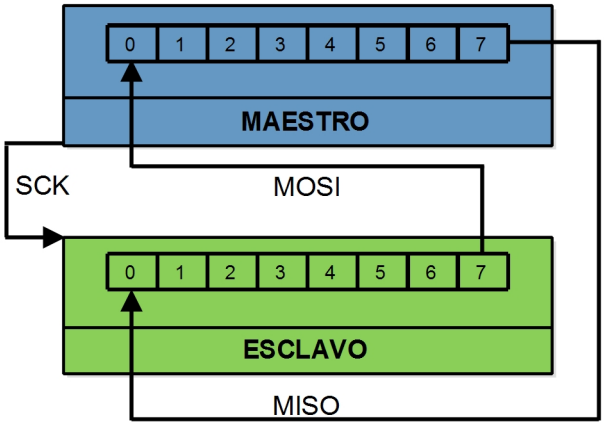

Figura 3.2: Comunicación entre dos dispositivos mediante SPI.

3.1.1 La comunicación SPI en el PIC18F4520

El PIC18F4520 cuenta con dos registros destinados a la transmisión de bytes mediante SPI. Estos son el registro SSPBUF (Serial Input Buffer) y el SSPSR (Shift Register). El primero es el registro de lectura y escritura donde cargar el byte a enviar y leer el byte recibido. El segundo tiene la función de, a partir del byte recogido del registro SSPBUF, transmitirlo por las vías MOSI o MISO[2], y cargar una vez finalizada la transmisión el nuevo dato recibido en el registro SSPBUF. Para avisar que el proceso se ha acabado se activan las banderas de interrupción junto con la señal de *Buffer Full*.

En el caso del PIC18F4520 los pines destinados a la comunicación por SPI son:

- MOSI - RC5/SDO (pin 24)
- MISO - RC4/SDI/SDA (pin 23)
- SCK - RC3/SCK/SCL (pin 18)

[2]La vía utilizada para esta transmisión depende de si el dispositivo está configurado como maestro o esclavo.

- NSS - RA5/NSS (pin 7)

Para acabar, el compilador MPLAB C18 de Microchip cuenta con la librería *spi.h* que ofrece toda una serie de funciones para trabajar con el SPI.

3.1.2 La comunicación SPI en la Pyboard

La Pyboard soporta también el estándar SPI. Para utilizarlo, además de establecer las conexiones necesarias, es necesario crear mediante código una instancia de la clase `pyb.SPI()` a la que se le pasarán todos los parámetros requeridos para establecer la comunicación con el otro dispositivo. Entre estos parámetros se encuentran el *baudrate*, cuál de los dos puertos SPI se desea utilizar o si el dispositivo actúa como esclavo o maestro. Una vez creada la instancia, ya se puede hacer uso de todas las funciones que ofrece el módulo para la comunicación vía SPI.

En el caso de la Pyboard los pines destinados a la comunicación por SPI son:

- MOSI - pin X8 o Y8
- MISO - pin X7 o Y7
- SCK - pin X6 o Y6
- NSS - pin X5 o Y6

Los pines que van de X8 a X5 pertencen al puerto SPI(1) mientras que los que van de Y8 a Y5 pertenecen al SPI(2). Cabe mencionar que no se pueden combinar las conexiones que ofrecen los dos puertos, sino que debe hacerse uso de todas las conexiones del mismo puerto.

3.2 El sistema operativo OSA-RTOS

El sistema operativo OSA-RTOS (http://pic24.ru) es un sistema operativo en tiempo real (RTOS) para microcontroladores PIC distribuido bajo la licencia *Berkeley Software Distribution* (BSD). Es gratuito y se puede encontrar y descargar fácilmente de internet. Su uso es compatible con la gran mayoría de compiladores para código escrito en C, entre ellos el C18, que es el utilizado en este proyecto, y su documentación puede encontrarse en la página web junto con varios ejemplos de su uso. Este se encarga de gestionar los recursos del sistema, además de programar la ejecución de las diversas tareas definidas en el programa en base a sus prioridades.

Para poder hacer uso del OSA-RTOS es necesario seguir los siguientes pasos, todos ellos explicados en la página web mencionada previamente:

- En primer lugar, descargar el archivo que contiene todos los ficheros necesarios [3].
- Después se debe añadir el fichero *osa.c* al proyecto en el que se quiere usar.

[3] Se puede descargar directamente de http://pic24.ru/doku.php/en/osa/ref/download/intro.

- En la carpeta donde se hayan ubicado los encabezados del proyecto se debe añadir o crear el fichero *OSAcfg.h*, donde se especifican, entre otras, el número de tareas que se quiere implementar. Este fichero puede estar en blanco pero siempre deberá estar presente.
- Añadir las líneas de código `include "OSA.h"` y `include "OSAcfg.h"` en todos los ficheros que vayan a utilizar los servicios ofrecidos por OSA-RTOS.
- Añadir una llamada a `OS_Init()` al inicio de la función `main()` del proyecto y otra a `OS_Run()` al final de la misma.

OSA-RTOS un sistema operativo cooperativo multitarea. Cooperativo significa que las tareas definidas por el usuario ceden voluntariamente los recursos que utilizaban o bien una vez han llegado al final de su ejecución o bien cuando están esperando a que otros recursos queden liberados para poder continuar. Para que una tarea libere los recursos que está utilizando debe existir dentro de la misma una llamada explicita a una de las muchas funciones que ceden el control de los recursos al sistema operativo. Multitarea quiere decir que el sistema operativo es capaz de gestionar y almacenar en memoria el estado de más de una tarea[4] de manera que puede ir cambiando de contexto a medida que se liberan recursos o se acaban tareas. Cada tarea es un hilo independiente de ejecución, con código propio y capacidad de comunicarse con el resto de tareas para acceder de forma ordenada a los recursos del sistema.

Una tarea en OSA-RTOS es una función en C que debe contener en su definición tanto un bucle infinito como una llamada a uno de los múltiples servicios que ofrece este sistema operativo para cambiar de contexto, de manera que se puedan liberar los recursos que se están utilizando. Dentro del bucle infinito es donde se debe colocar el cuerpo propio de la tarea que se quiere realizar.

Un ejemplo de tarea podría ser el siguiente:

```
// Ejemplo de tarea para OSA-RTOS
void my_Task(void)
{
   for(;;) // Bucle infinito
   {
      // Tarea que se desea que gestione OSA-RTOS
      OS_Yield(); // Dar paso a la siguiente tarea cediendo
                  // el control de los recursos al sistema operativo
   }
}
```

Figura 3.3: Ejemplo de tarea a gestionar por OSA-RTOS.

Para que el OSA-RTOS ejecute las tareas que se han definido se le debe indicar explícitamente la lista de tareas que debe gestionar. Para eso hay que añadir en el cuerpo de la función `main()`, entre las llamadas a `OS_Init()` y `OS_Run()`, la siguiente línea de código `OS_Task_Create(1,my_Task)` donde el primer argumento define la prioridad de la tarea mediante un número de cero a siete y el segundo argumento indica el nombre de la función que queremos definir como tarea para que

[4]Se refiere aquí al estado de la CPU y registros asociados.

el OSA-RTOS se encargue de su gestión. Se debe llamar a esta última función tantas veces como tareas se hayan definido. Cabe mencionar aquí que se deberá especificar también la creación de una nueva tarea en el fichero *OSAcfg.h*.

El ejemplo anterior quedaría entonces de la siguiente manera:

```
void main(void)
{
   // Inicio cuerpo principal
   // ...
   // Fin cuerpo principal

   // OSA-RTOS
   OS_Init();
   OS_Create_task(1,my_Task);
   // Resto de tareas
   OS_Run();
}
```

Figura 3.4: Código a añadir en la función principal del archivo donde se pretenda usar OSA-RTOS.

3.3 La comunicación por RF mediante el módulo nRF24L01+

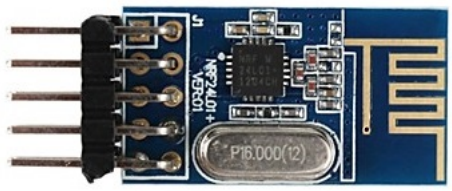

Figura 3.5: Vista del módulo nRF24L01+.

El objetivo del módulo nRF24L01+ es permitir establecer una vía de comunicación por RF entre microcontroladores. Desarrollado por *Nordic Semiconductor* trabaja en la banda ISM (*Industrial, Scientific and Medical*), entre los 2.4-2.4835GHz, a 3.3V y tiene un alcance máximo aproximado de 10 metros[5]. Su consumo oscila entre los 900nA (*Power Down Mode*) y los 13.5mA (ratio de envío de datos de 2Mbps). Consta de un oscilador de cristal, un sintetizador de frecuencia, un

[5]Basado en las pruebas realizadas por Pablo Sanz al desarrollar su librería.

amplificador de potencia y un modulador y demodulador que trabajan según el estándar GFSK (*Gaussian Frequency Shift Keying*).

Otra característica del nRF24L01+ es la posiblidad de, al configurarlo como receptor, activar una funcionalidad adicional que recibe el nombre de *MultiCeiver* y permite al módulo actuar como multireceptor recibiendo paquetes de hasta 6 emisores diferentes.

Además posee un sistema de preparado y lectura de paquetes llamado *Enhanced Shockburst* y otro de confirmación de recibo de paquetes llamado *Auto Acknowledgement*.

Para comunicarse con él es necesario establecer un canal de comunicación entre este y el microcontrolador del que recibe órdenes vía SPI.

3.3.1 Pines y conexiones

El nRF24L01+ consta de diez pines, cuatro para establecer las conexiones necesarias para habilitar la comunicación vía SPI con el microcontrolador al que se conecta, dos más para señales de control y cuatro para la alimentación, dos dedicadas al suministro de corriente y dos conectadas a tierra. De estas últimas solo es necesario hacer uso de una de cada.

Uso	**Nombre pin**	**Entrada/Salida**
SPI	MISO	Salida
	MOSI	Entrada
	SCK	Entrada
	NSS	Entrada
Control	CE	Entrada
	IRQ	Salida
Alimentación	VCC	Entrada
	GND	Salida

Tabla 3.1: Nombre y uso de los pines del módulo *nRF24L01+*

Las dos señales e control CE e IRQ, a pesar de no formar parte del estándar SPI, juegan un papel importante en la comunicación entre el microcontrolador y el transceptor nRF24L01+. La primera sirve para indicarle al nRF24L01+, dependiendo del modo de trabajo, que o bien inicie la transmisión de un paquete de datos o bien rastree el aire en busca de paquetes. La segunda se activa o cuando el nRF24L01+ ha recibido un paquete, si está en modo receptor, o cuando ha dejado de enviar, si está en modo emisor, .

Una vez conectado al microcontrolador y habilitado el SPI ya se puede empezar a hacer uso del módulo, teniendo en cuenta que el módulo nRF24L01+ siempre actúa como esclavo en su comunicación con el microcontrolador.

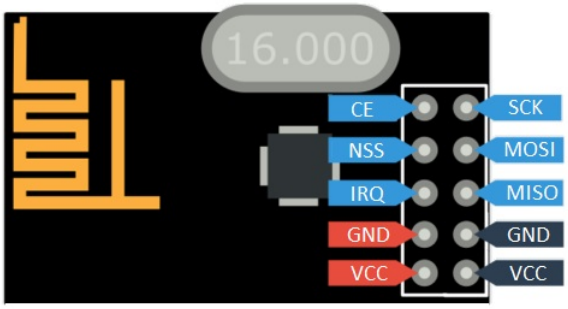

Figura 3.6: Distribución de pines del módulo nRF24L01+.

3.3.2 Configuración

El nRF24L01+ se configura mediante una serie de registros[6] de lectura y escritura cada uno de los cuales lleva asociado uno o más parámetros de configuración. En función de los valores de cada uno de estos registros el nRF24L01+ se comportará de una manera o de otra. Para modificar alguno de estos parámetros basta con sobrescribir el valor del registro asociado a ese parámetro. Para ello el nRF24L01+ cuenta con un juego de instrucciones de un byte que, transmitidas mediante SPI, permiten al microcontrolador dar órdenes al nRF24L01+.

Entre los parámetros que al usuario se le permite configurar cabe destacar el configurar el módulo como emisor o receptor, la frecuencia del canal de comunicación, la potencia de salida o el ratio de envío de datos en Mbps. Más estas no son, ni mucho menos, todas las opciones de configuración que ofrece el nRF24L01+.

3.3.3 Funcionalidades adicionales: *Enhanced ShockBurst*, *Auto Acknowledgement* y *MultiCeiver*

Con el objetivo de conseguir una comunicación mediante RF eficaz y fiable el nRF24L01+ dispone de una serie de funcionalidades adicionales implementadas con este fin.

En primer lugar, la tecnología *Enhanced ShockBurst* es la encargada de gestionar, de forma automática, el ensamblado y desensamblado de los paquetes que se enviarán por RF, su validación y de marcar los ciclos de la señal de reloj que se utiliza tanto para enviar paquetes de datos por parte del emisor como para rastrear el aire y demodular las señales encontradas en busca de un paquete válido por parte del receptor.

[6]Se pueden consultar todos ellos junto con su explicación en el *datasheet* del nRF24L01+ o en el Anexo A.

Figura 3.7: Estructura del paquete ensamblado según la tecnología *Enhanced ShockBurst.*

El Preámbulo, de un byte de longitud, está formado por la secuencia 01010101 o 10101010 y sirve para estabilizar y sintonizar el receptor. Después viene la *Address* que, con una longitud de tres a cinco bytes indica la dirección a qué esta destinada el paquete. En el caso de que el receptor que está demodulando la señal tenga configurada esa dirección como una de sus direcciones de recepción se seguirá leyendo el paquete. En caso contrario este se deshecha. De esta manera se garantiza que el paquete llega al destinatario correcto. A continuación encontramos el *Packet Control Field* donde, junto con otra información, se indica la longitud del paquete de datos que se está recibiendo. Tras este viene ya el *Payload*, de entre 1 y 32 bytes, que es propiamente el cuerpo del mensaje donde va contenida la información que se quiere transmitir. El paquete acaba con el campo dedicado al CRC (*Cyclic Redundancy Check*) de uno o dos bytes que sirve de código de detección de errores.

En segundo lugar, el mecanismo de*Auto Acknowledgement*, en el caso que este activado, consiste en el envío automático de un paquete por parte del receptor al emisor tras haber recibido y validado un paquete a modo de confirmación. Este mecanismo lleva asociada la funcionalidad de *Auto Retransmission*, activa sólo cuando la de *Auto Acknowledgement* lo está también. Esta, en el caso de que el emisor no reciba el paquete de *Auto Acknowledgement* a modo de confirmación dentro del margen de tiempo establecido por parte del receptor retransmite el paquete original hasta un máximo de 15 veces. Tanto el número de reintentos como el tiempo entre cada uno de ellos se pueden configurar dentro de unos rangos preestablecidos. El número de reintentos puede oscilar entre 0 y 15, mientras que el tiempo entre ellos puede variar desde los 250μS hasta los 4000μS en intervalos de 250μS.

En tercer y último lugar es posible, mediante la configuración del nRF24L01+ en modo receptor como *MultiCeiver*, que este actúe como hasta un máximo de seis receptores en uno. El nRF24L01+ dispone de seis *pipes* que se pueden activar o desactivar, cada uno de ellos actuando como un canal por el que se puede recibir información. Para activar un *pipe* es necesario habilitar dicho *pipe* en el registro EN_ADDR y asignarle una dirección. Por defecto están activados los *pipes* cero y uno[7]. Cada uno de estos *pipes* tiene asociada una dirección propia y diferente al resto de *pipes* y solo procesará los mensajes en cuyo campo *Adress* contengan la dirección propia del *pipe*. Sin embargo, las *pipes* dos a cinco comparten los cuatro bytes más significativos con la *pipe* uno, solo diferenciándose en el LSByte (*Least Significant Byte*).

[7]Nótese que los *pipes* van numeradas de cero a cinco.

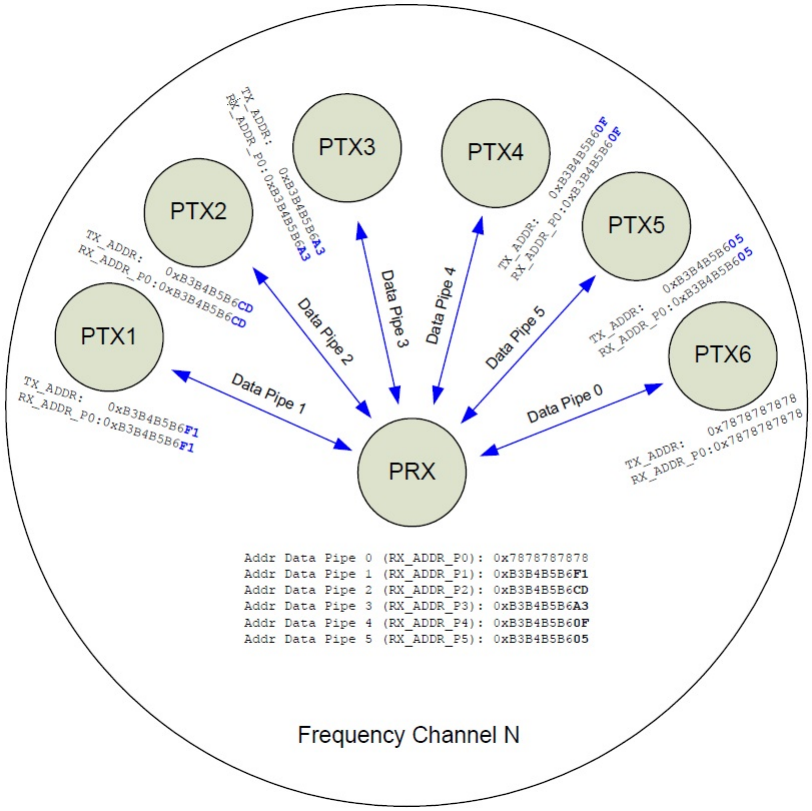

Figura 3.8: Ejemplo de funcionamiento del nRF24L01+ en modo *MultiCeiver*.

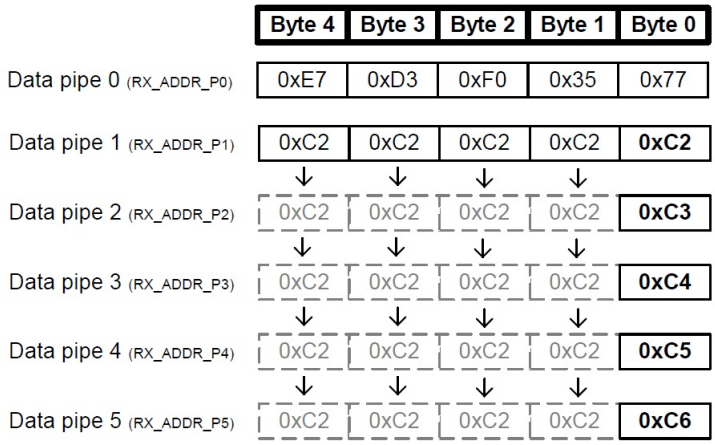

Figura 3.9: Ejemplo de una posible asignación de direcciones a las seis *pipes* del nRF24L01+.

Además, hasta que no se haya acabado de recibir un paquete por una *pipe* no se podrán recibir paquetes por el resto. Cabe destacar que todos los *pipes* configurados comparten el canal de Rf configurado, la velocidad de envío de datos Cada una de ellas dispone también de la opción de hacer uso del mecanismo de *Auto Acknowledge*. Todos los *pipes* tienen una serie de parámetros en común que son:

- La habilitación o deshabilitación del código CRC de detección de errores y, en el caso de su habilitación su longitud.
- La longitud de las direcciones.
- El canal de Radio Frecuencia.
- La velocidad de envío de datos.
- La ganancia del LNA.

La posibilidad de configurar un receptor como *MultiCeiver* pretende facilitar el trabajo a la hora de establecer redes de comunicación entre microcontroladores a través del nRF24L01+.

3.3.4 Modos de trabajo

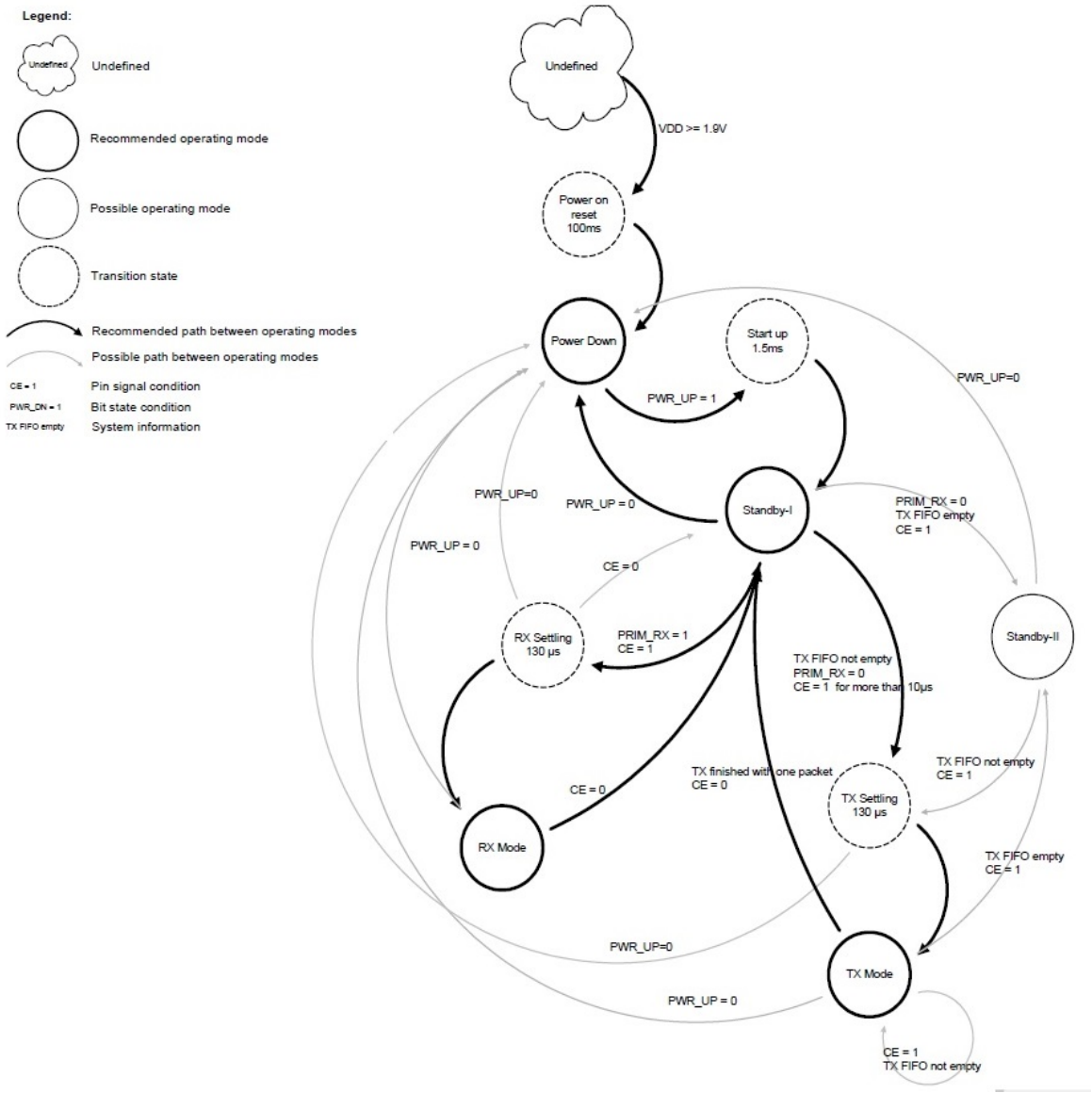

Figura 3.10: Diagrama de estados del nRF24L01+.

El nRF24L01+ presenta dos modos principales de trabajo. Estos son el modo emisor o TX y el modo receptor o RX. Estos se configuran mediante los bits PRIM_RX y PWR_UP del registro CONFIG.

Para configurar el nRF24L01+ como receptor se debe poner tanto el bit PRIM_RX como el bit

PWR_UP a uno. En este modo de trabajo es necesario además mantener el pin CE activado para rastrear el aire constantemente en busca de paquetes válidos. En el caso de encontrar alguno, este se carga en un hueco vacío de la RX FIFO, con capacidad de hasta 3 *payloads* de 32 bytes cada una. En el caso de no haber ningún huco disponible se descarta la *payload* contenida en el paquete. El módulo permanecerá en modo RX hasta que se reconfigure su modo de trabajo o se apague.

En el caso del emisor la configuración del bit PRIM_RX del registro CONFIG debe ser la opuesta al caso anterior, poniéndolo a 0. En cambio, el bit PWR_UP se debe poner a uno también. Además, debe tener cargada alguna *payload* para enviar en la TX FIFO, de igual capacidad que la RX FIFO. Este es el modo que permite al nRF24L01+ enviar paquetes vía RF. Para ello se debe activar la señal del pin CE durante al menos 10μS. En el caso de permanecer más tiempo la señal activa se irán enviando de forma consecutiva las *payloads* ubicadas en la TX FIFO hasta que esta se vacíe.

Además el nRF24L01+ presenta los modos de trabajo *Standby-I*, *Standby-II* y *Power Down* convenientemente descritos en el *datasheet*.

Mode	PWR_UP register	PRIM_RX register	CE input pin	FIFO state
RX mode	1	1	1	-
TX mode	1	0	1	Data in TX FIFOs. Will empty all levels in TX FIFOs[a].
TX mode	1	0	Minimum 10µs high pulse	Data in TX FIFOs.Will empty one level in TX FIFOs[b].
Standby-II	1	0	1	TX FIFO empty.
Standby-I	1	-	0	No ongoing packet transmission.
Power Down	0	-	-	-

Tabla 3.2: Modos de trabajo del nRF24L01+.

3.4 La librería UPC_nRF24L01

UPC_nRF24L01 es una librería desarrollada por Pablo Sanz en su PFC[6] para microcontroladores PIC18F4520 con el objetivo de facilitar la comunicación por RF mediante el módulo nRF24L01. Esta librería es la que se ha utilizado para implementar las rutinas de RF en los músicos, pues es compatible con el módulo nRF24L01+. Permite centrarse directamente en establecer una vía de comunicación por RF sin tener que preocuparse de las tareas de escritura y lectura de registros vía SPI. Pone a disposición del usuario funciones de más alto nivel encargadas de iniciar y configurar el SPI, los puertos del módulo nRF24L01+ y su definición como emisor o receptor, además del envío o recepción de información. En el caso de que se necesitara trabajar a más bajo nivel, escribiendo o leyendo directamente el valor de algún registro del módulo de RF, también incluye funciones que lo permiten.

Para poder hacer uso de ella se deben añadir al proyecto los ficheros *UPC_nRF24L01.c* y *UPC_nRF24l01.h* en los directorios de código fuente y encabezados respectivamente. Por último debemos incluir en el fichero que contiene el código principal del proyecto la línea `include "UPC_nRF24L01.h"`.

Nombre de la Función	Descripción
`SPI_Start`	Iniciar el hardware requerido para la comunicación vía SPI con el PIC18 como maestro.
`nRF24L01_Ports_Start`	Activar el resto de líneas necesarias para la comunicación con el módulo nRF24L01+.
`Start_TX_Mode_nRF24L01`	Iniciar y configurar el módulo nRF24L01 como dispositivo TX (emisor).
`Start_RX_Mode_nRF24L01`	Iniciar y configurar el módulo nRF24L01 como dispositivo RX (receptor).
`Send_Data_TX_Mode_nRF24L01`	Preparar el paquete a transmitir e iniciar el envío vía RF.
`Receive_Data_RX_Mode_nRF24L01`	Rastrear en busca de señales e informar del resultado y paquetes recibidos.
`Finish_nRF24L01_Operation`	Apagar el módulo nRF24L01 para reducir su consumo.
`Finish_SPI_Operation`	Desactivar el hardware asociado al SPI del PIC18.

Tabla 3.3: Funciones principales de la librería UPC_nRF24L01.

Para iniciar la transmisión o recepción de datos primero se ha de llamar a las funciones `SPI\_Start()` y `nRF24L01_Ports_Starts()`, puesto que estas se encargan de activar el módulo de RF e iniciar la comunicación entre el PIC y el nRF24L01+ vía SPI. Después se debe configurar el nRF24L01+ en modo emisor o receptor según la función que se desea que desempeñe. Una vez hecho esto ya se pueden utilizar las funciones de envío y recepción de datos. Por último se recomienda incluir una llamada a las dos funciones de finalización una vez enviados y recibidos todos los paquetes deseados.

3.5 El driver para Pyboard del módulo nRF24L01+

En el caso de la Pyboard se ha utilizado un driver[8] para comunicarse con el módulo nRF24L01+ desarrollado por Damien George. Este es de libre descarga y puede encontrarse en la página: https://github.com/micropython/micropython/blob/master/drivers/nrf24l01/nrf24l01.py junto con otro programa para probar y verificar el funcionamiento del driver.

Cómo la biblioteca desarrollada por Pablo Sanz, este programa facilita la comunicación vía SPI con el módulo de RF además de proveer al usuario de una serie de funciones para enviar y recibir paquetes que se encargan de gestionar la escritura y lectura de los registros y sus correspondientes bits necesaria para llevar a cabo la transmisión. Además de las funciones de envío y recibo contiene otras funciones que permiten gestionar los *pipes* habilitados y sus correspondientes direcciones.

La principal diferencia con la librería utilizada para los microcontroladores PIC18F4520 consiste en el método utilizado para saber si se ha recibido un paquete. En este caso no se avisa al

[8]Puede consultarse su código entero en el anexo B.

microcontrolador mediante la señal de interrupción IRQ si no que el microcontrolador, mediante la lectura del registro STATUS, es capaz de conocer el estado del nRF24L01+ y deducir si ha recibido un paquete.

El resto de diferencias observables a nivel de sintaxis o estructura del código son debidos, en gran medida, a los diferentes paradigmas de programación que siguen los lenguajes Python y C.

4 Hardware utilizado

4.1 Director

El director es, de todos los integrantes de la orquesta, el que más diferencias presenta respecto al resto. Los componentes que lo forman son una Pyboard programada en Python, una tarjeta de memoria microSD y un módulo de RF nRF24L01+.

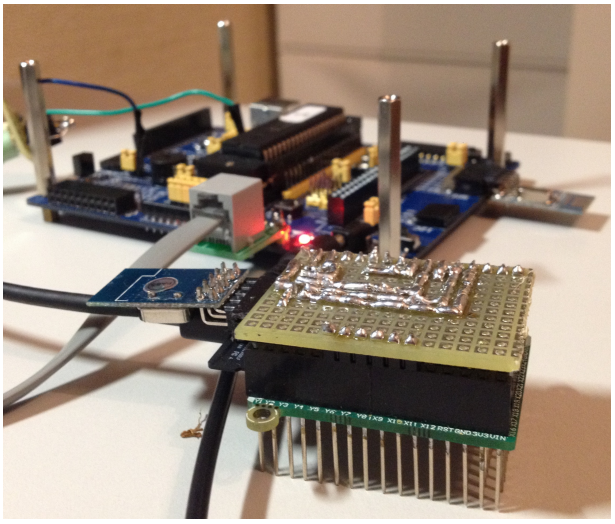

Figura 4.1: Aspecto final del director.

La Pyboard nace de la iniciativa en Kickstarter de Damien George, cuya idea era desarrollar una placa basada en un microcontrolador de 32 bits con la característica principal de que se pudiera programar en Python. Para ello Damien ha reescrito el lenguaje de programación Python (versión 3.4) y lo ha optimizado para que se ejecute de forma eficiente y consuma muy poca memoria RAM. Está versión de Python para la Pyboard incorpora un módulo al lenguaje, el módulo *pyb*. Este permite al usuario, a través de unas funciones y clases, controlar los distintos periféricos de la placa tales como el UART, I2C, SPI, ADC y DAC.

Las principales características del hardware de la placa son:

- Microcontrolador STM32F405RG
- 168 MHz Cortex M4 CPU con hardware de punto flotante
- 1024KiB de memoria flash ROM y 192KiB de memoria RAM
- Conexión micro USB para la alimentación y comunicación con el ordenador.
- Acelerómetro de 3 ejes (MMA7660)
- Posibilidad de conectar una tarjeta microSD
- 29 pines GPIO
- 4 LEDs, 1 botón de *reset* y otro botón a configurar por el usuario.
- 3 convertidores de analógico a digital de 12 bits y 2 convertidores de digital a analógico de 12 bits
- Regulador de voltaje de 3.3V integrado, capaz de suministrar 250mA con voltajes de entrada de entre 3.6V y 16V

Hay tres posibles maneras de controlar la Pyboard:

1. **REPL**: Conectando la placa al ordenador vía USB hace que el ordenador la detecte como un puerto virtual de comunicación al cual se puede asociar una terminal de Python. Desde la terminal se pueden ejecutar directamente comandos de Python, como se haría en un ordenador normal, con la diferencia de que estos serán serán procesados por la Pyboard.
2. **Archivo remoto**: Con la placa conectada al ordenador vía USB y habiendo asociado un terminal de Python a la Pyboard es posible indicar a esta última que ejecute programas guardados en el ordenador. Para ello, dentro del terminal se debe presionar la combinación de teclas `Ctrl-A` y tras ello escribir `python pyboard.py script_to_run.py` donde `script_to_run.py` es el programa que ejecutará la Pyboard.
3. **Archivo local**: Por último, ya sea gracias al pequeño sistema de gestión y almacenamiento de archivos incorporado en la memoria *flash* del microcontrolador o gracias a una tarjeta microSD que se puede conectar a la Pyboard es posible almacenar archivos y programas directamente en la Pyboard como se haría con cualquier dispositivo de almacenamiento. Al conectar la Pyboard al ordenador está aparece como un dispositivo de almacenamiento USB y se puede acceder y navegar a su memoria de esta manera. Si se copia un programa al sistema de archivos y se le llama `main.py` este se ejecutará directamente al alimentar la Pyboard. De esta manera se pueden ejecutar programas sin conectar la placa a un ordenador.

A la Pyboard se ha conectado el módulo nRF24L01+ mediante uno de los dos puertos SPI disponibles en la Pyboard. Para obtener un diseño final más compacto y práctico se han soldado en una placa de prototipado las conexiones necesarias entre la Pyboard y un conector en forma de codo hembra con 8 entradas donde se conecta finalmente el módulo nRF24L01+. Cabe destacar

de este microcontrolador que todos los pines que ofrecen una funcionalidad específica pueden ser utilizados también como pines GPIO. Su uso en una aplicación concreta se define mediante código, donde a la hora de programar un pin para utilizarlo se debe especificar su función[1].

Se ha elegido, de los dos puertos SPI que ofrece la Pyboard, el primero de ellos[2]. Este comprende los pines X8,X7,X6 y X5. Cada uno de estos, al usarse para establecer un canal de comunicación SPI, adquiere una función concreta. El pin X8 se conecta al pin MOSI del módulo de RF, el X7 al MISO, el X6 al SCK y el X5 al NSS Para la alimentación del módulo se ha decidido utilizar los pines 3V3 y GND más próximos al lugar por donde se conectará el módulo. Estos son los que se encuentran en la misma fila que los pines utilizados para las conexiones del SPI. Por último, el pin de la Pyboard que se ha conectado al pin CE del módulo nRF24L01+ ha sido el pin de la Pyboard numerado como X4. Nótese que en el caso del director no se ha hecho uso del pin de interrupción IRQ que ofrece el módulo de RF. Puesto que este módulo ofrece dos opciones a la hora de comprobar si se ha recibido un paquete, o bien lanzando una interrupción a través del pin IRQ o bien mediante la comprobación en el regitro STATUS, en el caso del director se ha optado por esta última con lo que no ha sido necesario conectar el pin IRQ al microcontrolador.

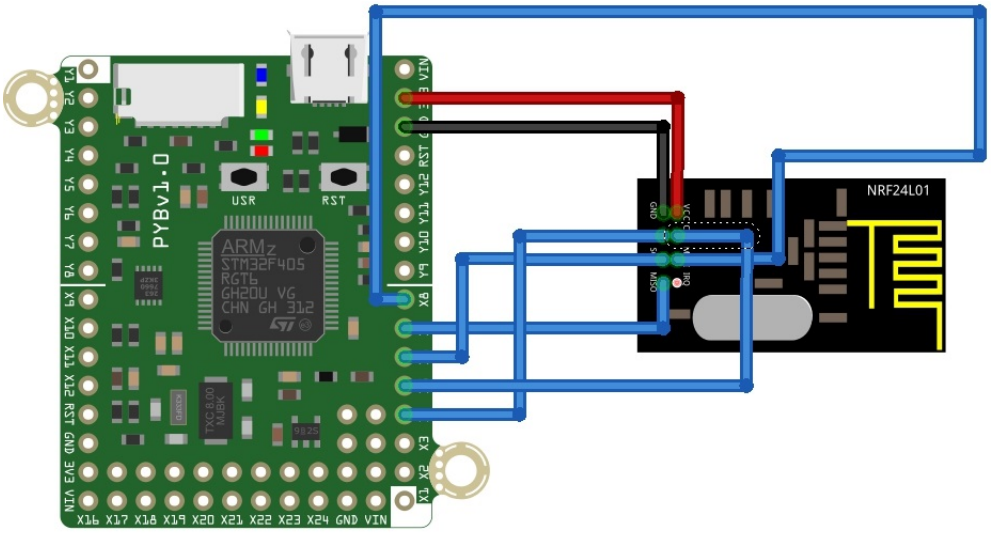

Figura 4.2: Conexiones entre la Pyboard y el módulo nRF24L01+.

La Pyboard ofrece además, de forma innata, una ranura para tarjetas microSD. En el caso de introducir una tarjeta, su papel es el de hacer de memoria ROM de la Pyboard y su contenido está siempre disponible para acceder mediante código. El uso de la tarjeta de memoria microSD en el caso del director es el de alamacenar las partituras de cada músico y el código que hace funcionar el director. Al ser la capacidad de la tarjeta de memoria varios órdenes de magnitud superior a la necesaria para almacenar las partituras y el código necesario se da por válido el argumento de que la orquesta no presenta ninguna limitación, por parte del director, sobre la longitud de la pieza a interpretar por la orquesta.

[1]Por defecto el pin será utilizado como GPIO.
[2]SPI(1) en el manual.

Por último, la Pyboard se alimenta directamente a través de la conexión USB micro-AB de que dispone.

4.2 Músicos

La única diferencia que presentan los músicos entre sí es el código que los hace funcionar. Todos ellos presentan los mismos componentes e idénticas conexiones entre ellos. Cada músico está formado por un microcontrolador PIC18F4520 montado sobre una placa de desarrollo Open18F4520. Esta placa, desarrollada por *WaveShare*, dispone de un gran número de puertos y circuitos integrados que facilitan enormemente la tarea de desarrollar prototipos de aplicaciones basadas en microcontroladores. Además, sigue permitiendo acceder a todos los pines de forma directa, con lo que no limita el uso del microcontrolador en el caso de que se quiera añadir alguna funcionalidad adicional. Se alimenta mediante conexión USB y pone en manos del usuario una interfaz que reduce en gran medida la dificultad de programar el microcontrolador.

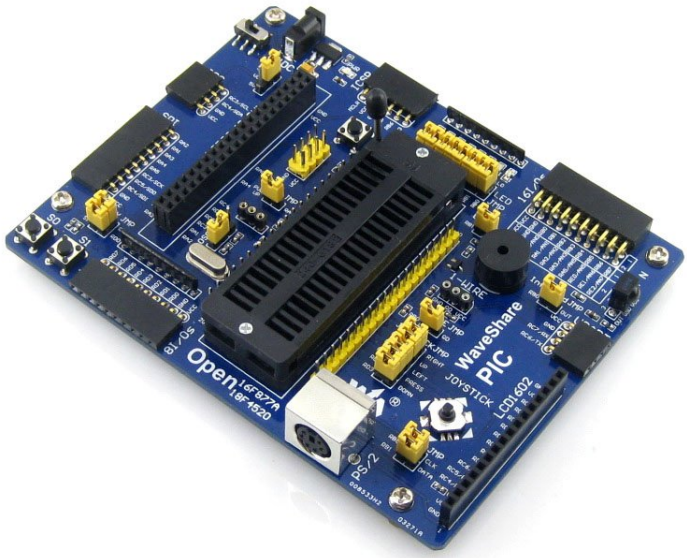

Figura 4.3: Placa de desarrollo Open18F4520.

Esta es la encargada de alimentar el PIC18F4520 y facilitar la señal de reloj, mediante un oscilador de 4MHz que lleva incorporado, que hace funcionar la CPU. Se ha aprovechado el puerto SPI integrado de que dispone para conectar el módulo de RF. En cambio el filtro RC se ha conectado directamente a los pines del microcontrolador. El terminal positivo se ha conectado al pin 36 y el negativo al pin 31, que actúa como toma de tierra. Este filtro es el encargado de convertir a analógica la señal digital proveniente de la salida PWM. Está compuesto por dos resistencias y un condensador y la señal filtrada pasa por una salida de audio en forma de jack de 3.5mm a la que se conectan los altavoces. La función de la primera resistencia, independiente

del filtro, es la de regular el volumen del sonido que producirán los altavoces ya que estos no ofrecen esa posibilidad.

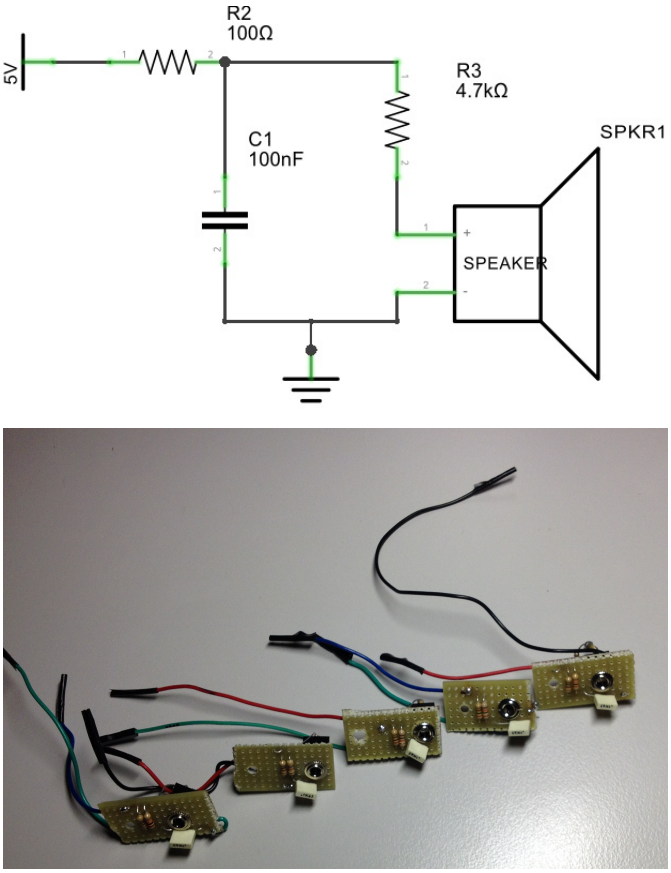

Figura 4.4: Esquemático del filtro RC con la salida de audio junto a los circuitos fabricados.

A continuación se citan algunas de las características principales del microcontrolador PIC18F4520:

Parámetro	**Valor**
Memoria de programa	32 KBytes
RAM	1536 Bytes
Número de pines	40
Rango de voltajes de funcionamiento	2-5.5 V
Frecuencia máxima de trabajo	40 MHz
Memoria EEPROM	256 Bytes
Periféricos de comunicación digital	1-UART, 1-A/E/USART, 1-SPI, 1-I2C-MSSP(SPI/I2C)
Puertos de entrada/salida	5: el A, B, C, D de 8 bits y el E de 4 bits

Tabla 4.1: Características principales del PIC18F4520.

5 Primera versión de la orquesta

Para la primera versión de la orquesta se ha escogido la aplicación de un planteamiento sencillo pero funcional que permita su posterior desarrollo hacia modelos más complejos y con menos limitaciones. A continuación se describe el proceso de desarrollo y el funcionamiento de esta primera aproximación, el código escrito con ese objetivo y el resultado final.

5.1 Diseño

En esta versión de la orquesta se han separado en tres fases, que después se han mantenido, las tareas a realizar desde que se encienden los músicos y el director hasta que los músicos finalizan la interpretación musical. Estas tres fases son:

1. Lectura de las partituras de cada músico y preparación de los mensajes con las notas e instrucciones por parte del director.

2. Establecimiento de la comunicación mediante RF e intercambio de información a través de los canales abiertos.

3. Interpretación de la pieza musical por parte de los músicos.

El director de la orquesta sólo interviene en las dos primeras fases, mientras que los músicos intervienen en las dos últimas. La única fase que tienen en común es aquella en la que músicos y director intercambian información entre ellos. Los programas, tanto para el director como los músicos se han desarrollado siguiendo este flujo de ejecución.

5.1.1 Flujo de ejecución

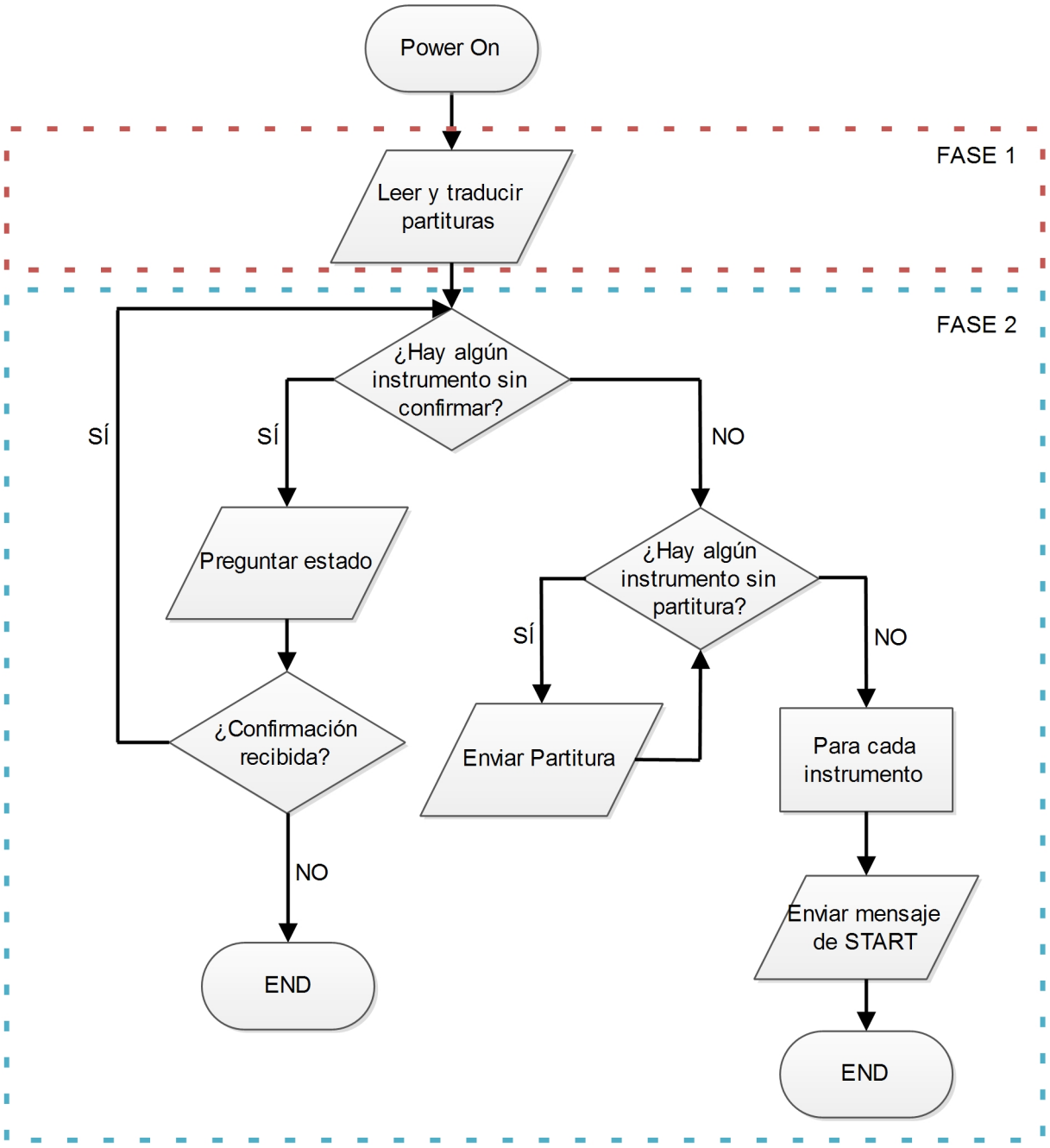

Figura 5.1: Flujo de ejecución del programa del director.

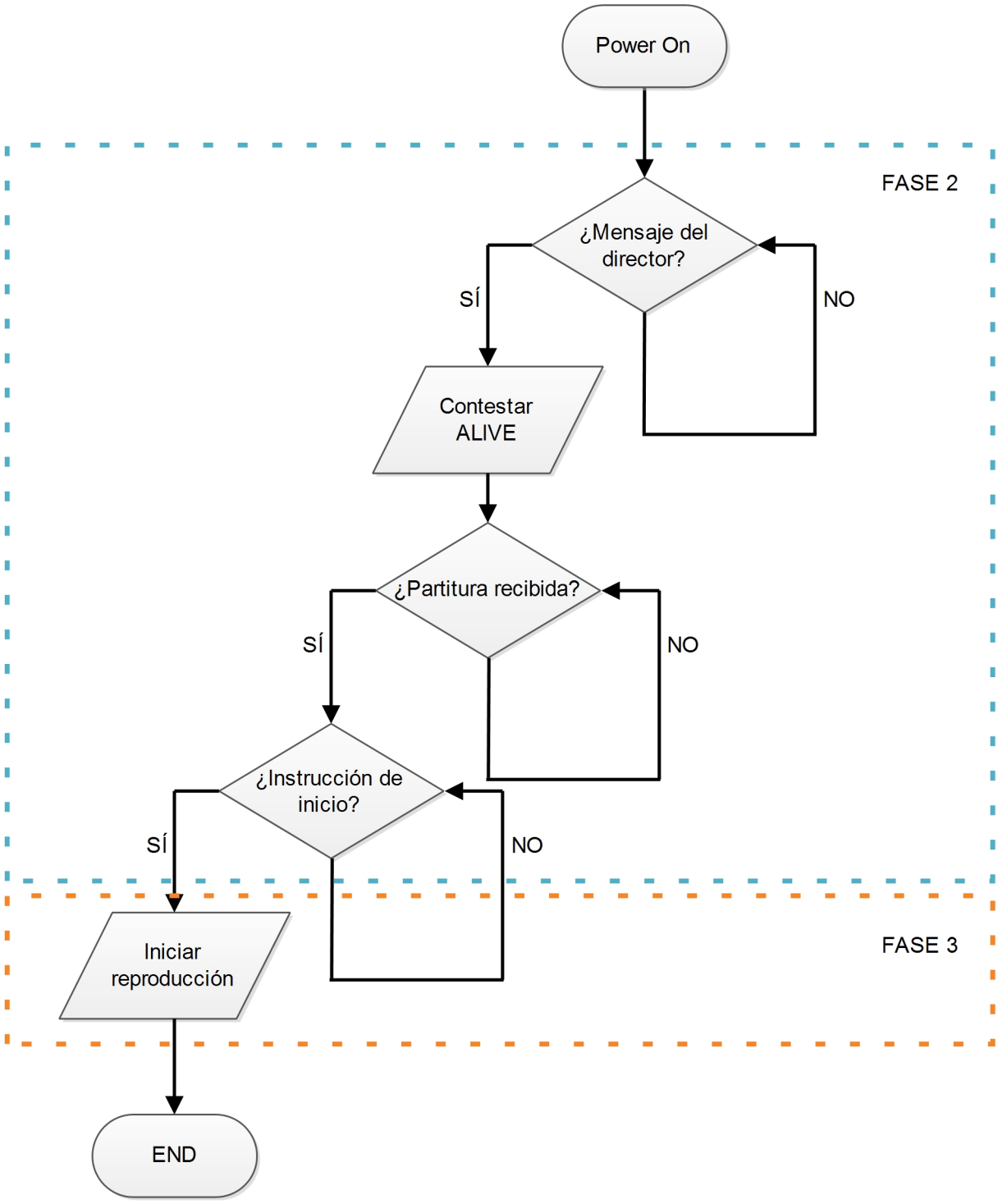

Figura 5.2: Flujo de ejecución del programa de los músicos.

El proceso seguido desde el encendido de los microcontroladores hasta el final de la interpretación puede deducirse de las figuras anteriores.

El proceso se inicia con el director leyendo de su memoria las partituras de cada músico y codificándolas de manera que o bien la instrucción o bien la nota y su duración ocupen un byte. Una vez ha procesado las partituras de cada músico procede a comprobar que todos los músicos se encuentren activos. Estos mientras se encuentran a la espera de recibir un mensaje del director preguntándoles por su estado. El director pregunta entonces uno a uno y a medida que van contestando que están activos va actualizando el valor del estado de los músicos en una base de datos muy simple que guarda en memoria. Cabe destacar aquí que si el director no es capaz de verificar que todos los músicos están activos, ya sea porque uno de los músicos no haya respondido o el mensaje de respuesta no haya sido recibido por el director se interrumpe la comunicación y no se continúa ejecutando el programa.

Una vez ha comprobado que todos los músicos están activos empieza a enviar, de forma secuencial, las partituras correspondientes a cada músico, que estos almacenarán en su memoria RAM. Concluida esta tarea y con cada músico sabiendo que ha de tocar el director envía entonces la orden a cada músico de que empiece a tocar. Para esta primera versión no se ha implementado ningún sistema de sincronización pero la velocidad de envío y el ínfimo lapso de tiempo que hay entre que el primer músico y el último reciben la instrucción de empezar a tocar hacen que el oído humano no sea capaz de detectar este desfase.

Por último, una vez los músicos han recibido la señal de inicio empiezan a procesar cada uno de los mensajes que han ido almacenando en RAM y de ellos extraen las notas que deben tocar y su duración. Con esta información generan la señal PWM que, una vez filtrada, hace sonar los altavoces.

5.1.2 Comunicación por Radio Frecuencia

Para que el intercambio de información se lleve a cabo de forma ordenada y evitando colisiones y referencias cruzadas entre los integrantes de la orquesta se ha decidido que el protocolo de comunicación entre microcontroladores sea siempre del tipo maestro-esclavo entre dos dispositivos. Además, en está aplicación concreta es más importante conseguir una alta fiabilidad a una alta velocidad. Al director se le ha asignado el papel de maestro y a los músicos el de esclavos. Esto implica que los músicos sólo pueden comunicarse con el director después de que este se haya puesto primero en contacto con ellos. Además se ha hecho uso de las tecnologías ofrecidas por el módulo nRF24L01+ de *Enhanced ShockBurst* para el ensamblado de mensajes y de *Auto Acknowledge* para la comprobación del recibo de mensajes.

Figura 5.3: Estructura de los mensajes utilizados para la primera versión de la orquesta.

Los mensajes intercambiados entre el director y los músicos presentan siempre una longitud

del tamaño de dos bytes. Estos dos bytes se colocan en el lugar de la *payload* y a ellos hay añadir el resto de bits que incorpora al paquete la tecnología *Enhanced ShockBurst*. Puesto que al configurar el módulo nRF24L01+ se debe indicar el tamaño de los mensajes que se van a enviar o recibir se ha decidido trabajar con la longitud más larga de las utilizadas y de esta manera no tener que reconfigurar el módulo en función del mensaje que se envía o se espera recibir. El primer byte siempre contiene un identificador que específica el tipo de mensaje y el segundo contiene el contenido propio del mensaje en caso de que se requiera alguno. Para está primera versión se han definido cuatro identificadores:

Identificador	**Valor en hexadecimal**	**Función**
`STATUS_ID`	0x01	Indica que el mensaje pregunta o responde al estado del microcontrolador
`NOTA_ID`	0x23	Indica que el contenido del mensaje es una nota o instrucción relacionada con la partitura
`TEMPO_ID`	0x48	Indica que el contenido del mensaje es el tempo de la partitura
`START_ID`	0x76	Indica al músico que inicie la ejecución

Tabla 5.1: Identificadores de los mensajes utilizados y sus respectivos valores y significados.

En esta primera versión la comunicación entre el director y los músicos se lleva a cabo mediante seis posibles mensajes:

Mensaje	**Emisor**	**Receptor**
`STATUS_ID + --`	Director	Músico
`STATUS_ID + ALIVE (0x98)`	Músico	Director
`TEMPO_ID + TEMPO`	Director	Músico
`NOTA_ID + NOTA`	Director	Músico
`NOTA_ID + INSTRUCCIÓN`	Director	Músico
`START_ID + --`	Director	Músico

Tabla 5.2: Mensajes utilizados en la primera versión de la orquesta.

Respecto al valor del segundo byte de los mensajes, en el caso de ser `--` implica que su valor es indiferente puesto que no se va a procesar. En el caso de la respuesta del músico a la pregunta del director sobre su estado se ha decidido que estos respondan con un mensaje cuyo contenido, además de especificar mediante el identificador que el mensaje es sobre su estado, contenga en el segundo byte el valor hexadecimal 0x98 al que se le ha asociado el significado de indicar al director que el músico en cuestión está activo. En el caso del `TEMPO`, la `NOTA` y la `INSTRUCCIÓN` su valor varía en función de la partitura y la nota a tratar en concreto. Las instrucciones se envían bajo el identificador `NOTA_ID` puesto que son instrucciones referentes a la partitura.

El hecho de trabajar con la información a dos niveles facilita mucho la tarea de procesar e interpretar los mensajes y da mucha flexibilidad para implementar nuevos mensajes.

Otro tema que se ha tratado es el de asignar a cada músico y al director una dirección de RF

de cinco bytes única a modo de identificador y especificar así el destinatario de cada mensaje. Los músicos funcionan con dos direcciones, la suya propia de recepción para distinguir de todos los mensajes aquellos que van destinados a ellos y una de emisión asociada al director que es a la que envían los mensajes dirigidos al director. El director tiene asociada también una dirección de recepción pero, a diferencia de los músicos, cuatro direcciones de emisión. Estas cuatro direcciones de emisión son cada una de las direcciones de recepción de los músicos. Las direcciones que se han elegido son las siguientes:

Rol orquesta	**Dirección Recepción**	**Dirección Emisión**
Músico - Violín	0x41, 0x41, 0x41, 0x41, 0x41	0xB2, 0xB2, 0xB3 0xB4 0x01
Músico - Bajo	0x11, 0x11, 0x11, 0x11, 0x11	0xB2, 0xB2, 0xB3 0xB4 0x01
Músico - Guitarra 1	0x71, 0x71, 0x71, 0x71, 0x71	0xB2, 0xB2, 0xB3 0xB4 0x01
Músico - Guitarra 2	0xD1, 0xD1, 0xD1, 0xD1, 0xD1	0xB2, 0xB2, 0xB3 0xB4 0x01
Director	0xB2, 0xB2, 0xB3 0xB4 0x01	0x41, 0x41, 0x41, 0x41, 0x41 0x11, 0x11, 0x11, 0x11, 0x11 0x71, 0x71, 0x71, 0x71, 0x71 0xD1, 0xD1, 0xD1, 0xD1, 0xD1

Tabla 5.3: Direcciones de emisión y recepción de los distintos integrantes de la orquesta.

Esta dirección es la que la funcionalidad *Enhanced ShockBurst* añade al ensamblar el mensaje en la parte de la dirección. Para ello, las direcciones correspondientes deben estar cargadas en los registros del módulo nRF24L01+ destinados a tal uso: el registro TX_ADDR debe contener la dirección de destino en el caso del emisor del mensaje y el registro RX_ADDR_P0 debe contener la propia dirección de recepción en el caso del receptor. Aunque se podría utlizar cualquiera de las 6 *pipes* disponibles en el nRF24L01+ se ha decidido utilizar para este proyecto la *pipe* 0 ya que por defecto tiene activada la funcionalidad *Auto Acknowledge*. El resto de parámetros de RF utilizados para establecer la comunicación han sido:

- Frecuencia del canal de comunicación de 2.464±1 GHz, el equivalente a cargar el valor decimal 64 en el registro destinado al canal de RF.

- Velocidad de transmisión de 1 Mbps.

- Intervalo de 4750 μS entre reenvios de mensajes hasta un máximo de 15 intentos.

- Potencia de transmisión de 0 dBm.

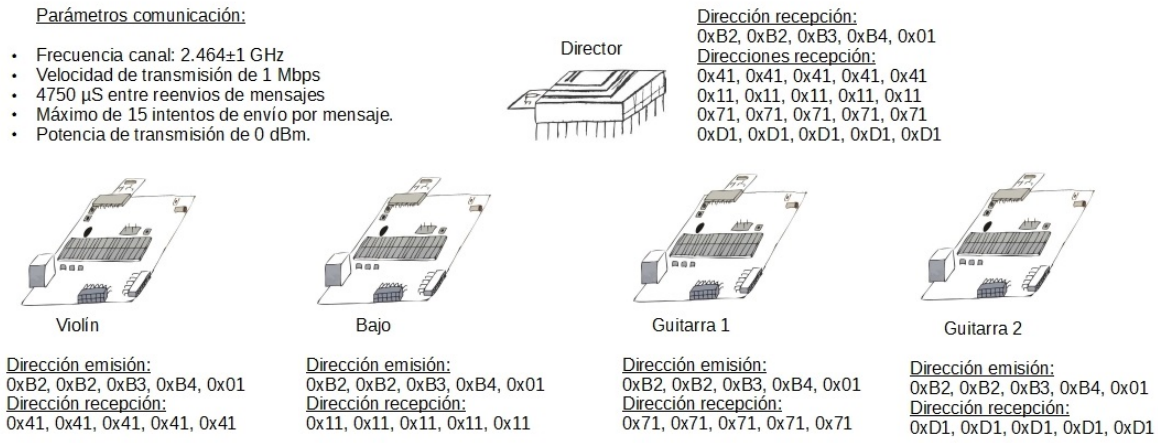

Figura 5.4: Configuración de los parámetros de comunicación por RF para la primera versión de la orquesta.

5.1.3 Formato de las partituras

Sin ser imprescindible para el proyecto hay que mencionar brevemente de que modo se han almacenado las partituras en la memoria del director ya que este espera encontrárselas con un formato concreto que es el que sabe procesar.

Las partituras se han almacenado en una carpeta que se ha creado en la tarjeta microSD insertada al director. En su interior se ha separado la partitura de cada músico en un fichero de texto plano y dentro de cada fichero se ha dedicado una línea a cada nota o instrucción. Las notas[1] siguen el formato `play(nota,duración)` seguido de un salto de línea. En cuanto a las instrucciones que se pueden encontrar, estás son:

- `repeatmarker()` y `repeat(número)`: la primera indica el punto en la partitura desde el que se debe empezar a repetir y la segunda traslada el puntero de la nota actual a la línea de inicio de repetición, además de indicar el número de veces que se debe repetir ese fragmento.
- `setbase(número)`: indica la octava de base de la canción.
- `stop()`: indica que se ha llegado al final de la partitura.
- `pause()`: indica una pausa y su duración.

Mediante la combinación de las instrucciones previamente mencionadas y las notas que componen la melodía se han escrito las partituras de los cuatro músicos.

[1] Estas están escritas según la convención inglesa de a,b,c,d,e,f,g.

5.1.4 Programa del director

Para realizar y dar soporte a todo el planteamiento explicado en el apartado anterior se ha tenido que desarrollar y adaptar el código necesario. En el caso del director este se ha estructurado en tres módulos distintos separados según las funciones y tareas a que está destinado cada uno. Un primer módulo, *note_cipher.py* es el responsable de la lectura de las partituras y su traducción a un valor numérico del tamaño de un byte. Un segundo módulo, llamado *nrf24l01.py* es el encargado de gestionar las comunicaciones vía SPI con el nRF24L01+ y del envío y recepción de mensajes por radiofrecuencia. Por último, el módulo *director.py* es el que conoce el funcionamiento de la orquesta y las tareas que se han de realizar y, a partir de los módulos anteriores es capaz de llevarlas a cabo.

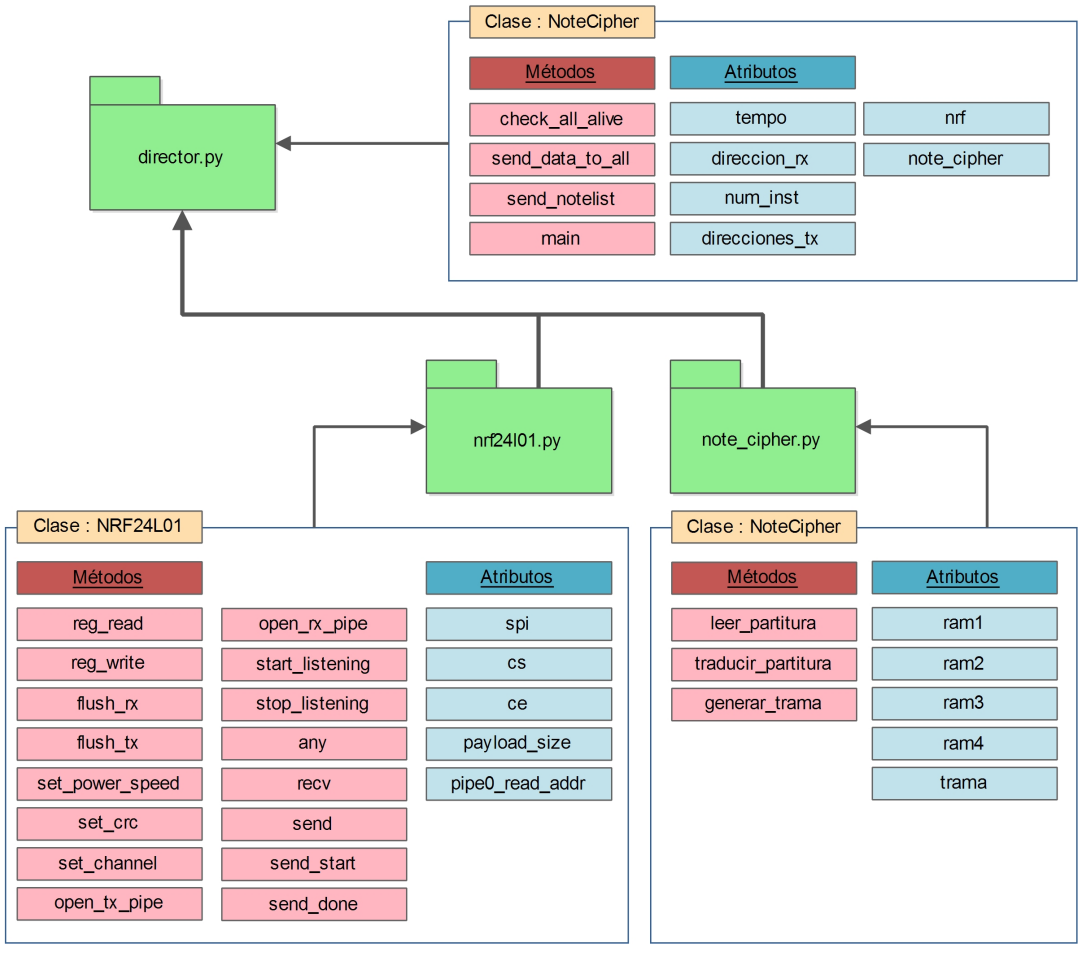

Figura 5.5: Estructura del programa desarrollado para el director en la primera versión.

5.1.4.1 Módulo note_cipher.py

El módulo *note_cipher.py*[2] está compuesto por la clase `NoteCipher`. Esta clase contiene cuatro atributos y tres métodos, además del constructor, que son los encargados de leer, procesar y generar la partitura al formato que enviará el director por RF.

La generación de las partituras se realiza de forma inmediata al instanciar la clase `NoteCipher` ya que el constructor se ha definido de la siguiente forma:

```
def __init__(self, part1, part2, part3, part4):

    pyb.LED(4).on()

    # guardamos en RAM las tramas de los instrumentos
    # a partir de la lectura del fichero correspondiente

    self.ram1 = self.traducir_partitura(self.leer_partitura(part1))
    self.ram2 = self.traducir_partitura(self.leer_partitura(part2))
    self.ram3 = self.traducir_partitura(self.leer_partitura(part3))
    self.ram4 = self.traducir_partitura(self.leer_partitura(part4))

    self.trama = self.generar_trama()

    pyb.LED(4).off()
```

Los atributos propios de la clase se definen a partir de los resultados de los métodos de la clase. Primero se generan las partituras de cada instrumento y se almacenan cada una en una lista en memoria. Una vez procesadas se genera la trama, que es una lista de listas tal que:

```
[[note 0 - musician 0, note 0 - musician 1, note 0 - musician 2, note 0 - musician 3 ],
 [note 1 - musician 0, note 1 - musician 1, note 1 - musician 2, note 1 - musician 3 ],
 ,...,
 [note n - musician 0, note n - musician 1, note n - musician 2, note n - musician 3 ]]
```

La longitud de la lista corresponde a la longitud de la partitura más larga y la diferencia entre partituras se compensa completando el número de notas de diferencia con ceros. Esta lista la genera el siguiente método:

```
def generar_trama(self):

    max_length = max([len(p) for p in [self.ram1,self.ram2,self.ram3,self.ram4]])

    equal_lengths = [p+[0]*(max_length-len(p)) for p in
        [self.ram1,self.ram2,self.ram3,self.ram4]]

    return [[equal_lengths[0][i],equal_lengths[1][i],equal_lengths[2][i],
             equal_lengths[3][i]] for i in range(max_length)]
```

[2]Su código completo puede consultarse en el Anexo B.

De este modo, para acceder a las partituras de la pieza musical solo se ha de acceder al atributo `trama` de la instancia de la clase `NoteCipher`.

5.1.4.2 Driver nrf24l01.py

El módulo *nrf24l01.py*[3], como se ha mencionado previamente, es el encargado de las comunicaciones por RF y con módulo nRF24L01+. Este consta también de una clase, la clase `NRF24L01` que hay que instanciar para poder hacer uso de sus métodos. Al instanciarla se deben definir las líneas con las que se va a establecer comunicación con el nRF24L01+ mediante SPI especificando cual de los dos buses SPI se va a utilizar y los pines que se utilizarán como líneas de CE y NSS. Respecto a la Radio Frecuencia hay que explicitar el canal que se desea utilizar y el tamaño en bytes de los mensajes que se van a enviar o se espera recibir. El resto de la configuración hay que hacerla de forma manual. Para ello esta clase dispone de métodos para escribir y leer los registros del nRF24L01+. En su inicialización realiza además una comprobación de la conexión con el nRF24L01+, lanzando un error si no consigue comunicarse.

```
    # set address width to 5 bytes and check for device present
    self.reg_write(SETUP_AW, 0b11)
    if self.reg_read(SETUP_AW) != 0b11:
        raise OSError("nRF24L01+ Hardware not responding")
```

Figura 5.6: Fragmento del constructor que realiza la comprobación de comunicación con el nRF24L01+.

Una característica importante de este *driver* del módulo de RF para la Pyboard es el planteamiento de recepción de mensajes. Este no hace uso de la interrupción facilitada por la línea de IRQ, sino que detecta si se ha recibido algún paquete a partir de la lectura del bit RX_EMPTY del registro FIFO_STATUS. Esto resulta en que en el módulo principal se debe programar un bucle de lectura de este bit en el momento en que se espera recibir un mensaje para determinar si se ha recibido.

```
def any(self):
    return not bool(self.reg_read(FIFO_STATUS) & RX_EMPTY)
```

Figura 5.7: Método para determinar si se ha recibido un mensaje.

Por lo general se ha respetado el código original del módulo, pero se ha tenido que modificar la rutina que habilita los *pipes* de recepción. El nRF24L01+, por especificaciones de diseño, permite su uso como multireceptor. Se puede usar también como multiemisor, pero entonces la transferencia del mensaje se hace mediante *broadcast*, como si se tratara de una emisora de radio, y se pierde la posibilidad de usar la funcionalidad de *Auto Acknowledge* ya que entonces todos los receptores deberían compartir la misma dirección de recepción de mensajes. Se ha probado de implementar esta posibilidad de *broadcasting* para transmitir un mismo mensaje a

[3]Su código completo puede consultarse en el Anexo B.

todos los músicos, pero la tasa de errores era superior a la permisible para garantizar el correcto funcionamiento de la orquesta. Por ello se ha optado por establecer siempre una comunicación dirigida entre dos microcontroladores con la tecnología *Auto Acknowledge* activada utilizando siempre la *pipe* 0. Para ello se ha debido modificar parte del código. Como la dirección asignada a la *pipe* 0 va cambiando en función del músico del que se espera recibir un mensaje se debe también actualizar la dirección de emisión para garantizar que el envío del mensaje de *Auto Acknowledge* se envía a la dirección correcta.

La modificación realizada consiste en verificar si la *pipe* que queremos habilitar tiene activada la funcionalidad de *Auto Acknowledge* y, en el caso de que así sea, modificar la dirección de emisión para que cuadre con la de recepción.

```
def open_rx_pipe(self, pipe_id, address):
   assert len(address) == 5
   assert 0 <= pipe_id <= 5
   if pipe_id == 0:
      self.pipe0_read_addr = address
   if pipe_id < 2:
      self.reg_write(RX_ADDR_P0 + pipe_id, address)
   else:
      self.reg_write(RX_ADDR_P0 + pipe_id,address[0])
   self.reg_write(RX_PW_P0 + pipe_id, self.payload_size)
   self.reg_write(EN_RXADDR, self.reg_read(EN_RXADDR) | (1 << pipe_id))
   # Code added for proper usage of Auto Acknowledge
   if self.reg_read(EN_AA) & (1<<pipe_id) == (1<<pipe_id):
      self.reg_write(TX_ADDR,address)
```

Figura 5.8: Modificaciones realizadas a la rutina de habilitar *pipes* de recepción para poder usar la tecnología de *Auto Acknowledge*.

5.1.4.3 Módulo director.py

Este módulo es el módulo principal[4] del director, al que están orientados los dos citados anteriormente. La clase `Conductor` esta dotada de dos atributos que son instancias de las clases contenidas en *note_cipher.py* y en *nrf24l01.py* y que permiten acceder así a todos sus métodos y atributos.

El método `main()` de la clase `Conductor` es el que contiene todas las tareas que corresponden al director y que se deben realizar por su parte para el funcionamiento de la orquesta.

[4]Se puede consultar su código entero en el Anexo B.

```
def main(self):

    while not self.check_all_alive():
      pass
    pyb.udelay(40)
    self.send_data_to_all(TEMPO_ID,self.tempo)
    pyb.udelay(40)
    for instrument_tx in self.direcciones_tx:
      self.send_notelist(instrument_tx)
    pyb.udelay(40)
    self.send_data_to_all(START_ID,0x00)
```

Figura 5.9: Método principal del módulo *director.py*

Se puede observar como la primera tarea que se lleva a cabo es la comprobación de que todos los músicos están activos. En el caso de que el director no pudiera confirmar que todos ellos están activos no saldría del bucle y el programa no seguiría con su ejecución. Una vez el director sabe que todos los instrumentos están activos envía a cada uno de ellos el tempo correspondiente a la pieza que se va a reproducir. El método `send_data_to_all` requiere de dos parámetros, donde el primer parámetro será el primer byte del mensaje enviado y el segundo parámetro será el segundo byte del mismo mensaje. Este método realiza los envíos de forma secuencial, a un músico detrás de otro. Tras el envío del tempo se envía la partitura correspondiente a cada músico mediante un método específico y por último se manda la orden de iniciar la reproducción.

5.1.5 Programa de los músicos

El código de los músicos[5], como se ha explicado en los antecedentes de este proyecto, se ha basado en parte el trabajo realizado por Victor Timofeev consistente en el desarrollo de un cuarteto musical en un microcontrolador PIC mediante el uso del sistema operativo para microcontroladores también desarrollado por él, OSA-RTOS. Se ha utilizado además el trabajo realizado por Joan Calvet en su PFC adaptándolo para su funcionamiento por radiofrecuencia y la librería UPC_nRF24L01 desarrollada por Pablo Sanz.

5.1.5.1 Aspectos generales

El programa se ha planteado de la forma más genérica posible con el objetivo de reutilizar el máximo código posible de un músico a otro. Para ello se han definido un archivo de encabezados con el nombre*Config.h* que permite seleccionar, mediante directivas de preprocesado, el instrumento de cada músico descomentando su línea y dejando las otras tres comentadas. Una vez elegido el instrumento, existen otras directivas de preprocesado en el archivo principal (*quartet_main.c*) que resultan en una compilación condicional que incluirá en el archivo compilado las especificaciones propias del instrumento definido y dejará fuera las demás.

Un ejemplo del uso que se ha dado a este planteamiento es el modo de asignar el valor de la

[5]Puede consultarse el archivo principal, *quartet_main.c* en el Anexo B.

variable `direccion_rx`. Esta contiene la dirección de recepción de mensajes por Radio Frecuencia del músico y varía de un instrumento a otro. También se han generalizado las tareas que gestiona el sistema operativo OSA-RTOS para que funcionaran con independencia del instrumento.

Además, se han tenido que modificar los bits de configuración del archivo *ConfigBits.h* para adaptarlo al microcontrolador PIC18F4520 y al uso del bus SPI. La modificación más importante es la de trabajar con un oscilador a 4MHz del que se ha activado el PLL mediante la linea de configuración `#pragma config OSC = HSPLL`. Se ha decidido trabajar a 16 MHz, una frecuencia suficientemente alta para que el procesado y enlazado de las distintas notas de la partitura se realice de manera fluida y la comunicación por SPI con el módulo nRF24L01+ no de problemas. Al variar la frecuencia de trabajo se ha tenido que modificar también el valor del PR2 para mantener el valor original del periodo del TIMER2 que es el que se utiliza para generar la señal PWM. El valor que se ha dado al PR2 es el de `PR2 = 51-1`.

```
//#define Bass
//#define Violin
//#define Guitar1
#define Guitar2

#if defined(Bass)&& defined(Violin)
   #error "Only one instrument allowed"
#endif

#if defined(Bass)&& defined(Guitar1)
   #error "Only one instrument allowed"
#endif

#if defined(Guitar1)&& defined(Violin)
   #error "Only one instrument allowed"
#endif
```

Figura 5.10: Encabezados que permiten generalizar el código de los músicos.

```
#ifdef Bass
   unsigned char direccion_rx[5]={0x11, 0x11, 0x11, 0x11, 0x11};
#endif
#ifdef Violin
   unsigned char direccion_rx[5]={0x41, 0x41, 0x41, 0x41, 0x41};
#endif
#ifdef Guitar1
   unsigned char direccion_rx[5]={0x71, 0x71, 0x71, 0x71, 0x71};
#endif
#ifdef Guitar2
   unsigned char direccion_rx[5]={0xD1, 0xD1, 0xD1, 0xD1, 0xD1};
#endif
```

Figura 5.11: Ejemplo de directivas de preprocesado del archivo principal.

Otro cambio que se ha hecho antes de entrar en la inclusión de las rutinas de radiofrecuencia ha sido la modificación de parte del código de la librería UPPC_nRF24L01 con el mismo objetivo, explicado previamente, que se ha modificado el módulo *nrf24l01.py* del director. Esta modificación se ha realizado dentro de la función `Start_RX_Mode_nRF24L01`, en las líneas 4 y 5 de código de la siguiente figura:

```
Write_nRF24L01_Register (SETUP_AW, TX_RX_Address_Width);                    // Set
    Width for all Addresses: 0b01 => 3 bytes, 0b10 => 4 bytes, 0b11 => 5 bytes
Write_nRF24L01_Register (RF_CH, Frequency_Channel);                          // Set
    the frequency channel nRF24L01 operates on
Write_nRF24L01_Register (RF_SETUP, RF_Data_Rate*0b1000 + RF_Output_Power*0b10 +
    LNA_Gain);  // Set nRF24L01 RF Data Rate, Power and LNA Gain

if (Read_nRF24L01_Register(EN_AA)&&0b00000001==0b00000001)
   Write_nRF24L01_Address_Register (TX_RX_Address_Width, TX_ADDR, RX_Pipe0_Address);

Write_nRF24L01_Address_Register (TX_RX_Address_Width, RX_ADDR_P0, RX_Pipe0_Address);
    // Define RX Address for Pipe0 (3, 4 or 5 bytes)
Write_nRF24L01_Address_Register (TX_RX_Address_Width, RX_ADDR_P1, RX_Pipe1_Address);
    // Define RX Address for Pipe1 (3, 4 or 5 bytes)
```

Figura 5.12: Fragmento modificado de la rutina de configuración como receptor de RF de la librería UPC_nRF24L01.

5.1.5.2 Recepción de mensajes por RF

En cuanto a las rutinas de Radio Frecuencia, estás se han incluido dentro la función `main` del archivo *quartet_main.c*. Su principal tarea es no dejar de rastrear el aire en busca de mensajes dirigidos al músico y procesarlos según los identificadores hasta recibir la instrucción de iniciar la melodía.

Lo primero que se ha hecho es iniciar el bus SPI, el módulo nRF24L01+ y configurar este último como receptor. Después se entra en un primer bucle infinito de escucha, en el que se espera recibir la pregunta del director por el estado del músico. Una vez recibido este mensaje se cambia el modo del nRF24L01+ a emisor y se contesta al director conforme el músico está activo. El hecho de recibir este mensaje provoca la salida del bucle y el inicio de la siguiente fase. En el caso de que no se recibiera ningún mensaje del director el músico se quedaría indefinidamente ejecutando el contenido de este bucle. En el caso de los músicos la comprobación de si ha llegado un mensaje si que se hace mediante la interrupción IRQ. Nótese también como cada vez que se finaliza una operación con el módulo de RF se incluye en el código una llamada a la función `Finish_nRF24L01_Operation`, que se encarga de poner el nRF24L01+ en modo de bajo consumo mientras no se vuelva a requerir su uso.

```
Init();       // Init periphery
InitSoundVariable(&S, cola_rx);
SPI_Start(0b10);
nRF24L01_Ports_Start();
i=0;
while(1)
{
   Start_RX_Mode_nRF24L01(0b11, 0b1000000, 0, 0b11, 1, 1, direccion_rx, 0x00, 0x00,
       0x00, 0x00, 0x00, 0, 2, 0, 0, 0, 0, 0);
   informe=Receive_Data_RX_Mode_nRF24L01(0, 100, 2, mensaje_llegada); //No Checksum
   Finish_nRF24L01_Operation();
   if(!(informe & 0b00010000))
   {
      if(mensaje_llegada[0]==STATUS_ID)
      {
         mensaje_rebotado[0]=STATUS_ID;
         mensaje_rebotado[1]=ALIVE;
         Start_TX_Mode_nRF24L01(0b11, 0b1000000, 0, 0b11, 1, 1, 0b0011, 15, 0, 2);
         Delay100TCYx(100);
         Send_Data_TX_Mode_nRF24L01(0,0b11,direccion_tx,2,mensaje_rebotado);
         Finish_nRF24L01_Operation();
         break;
      }
   }
}
```

Figura 5.13: Fragmento de la función principal donde puede apreciarse la inicialización del SPI y el módulo de RF, así como el bucle de espera al mensaje del director.

Después de este primer establecimiento de comunicación entre músico y director el músico se prepara para empezar a recibir instrucciones del director. Para ello se vuelve configurar el módulo de RF como receptor y se entra en el segundo bucle infinito de escucha, y cuando se recibe un mensaje, tras determinar el identificador que contiene, se actúa en consecuencia. Si el mensaje recibido incluye el identificador `TEMPO_ID` el músico almacena el segundo byte del mensaje en una variable `tempo` que será la que servirá para marcar la velocidad de la interpretación. Si, en cambio, el mensaje recibido incluye el identificador `NOTA_ID`[6] se almacena en una cola de reproducción el valor que le acompaña. Por último, si el identificador del mensaje es `START_ID` entonces se provoca la salida de este segundo bucle y se pasa acto seguido a iniciar la interpretación de la pieza musical.

Es aquí donde entra en juego el sistema operativo en tiempo real OSA-RTOS. Las tareas que este se encarga de gestionar son dos:

- `Task_INS`, de prioridad 0, es la que o bien genera o bien procesa, a partir de la cola de reproducción, la señal PWM que produzca el sonido acorde a la nota tratada o la instrucción interpretando su significado y llevándola a cabo.
- `Task_CONDUCTOR`, de prioridad más baja 1, es la encargada de marcar el inicio y fin de la re-

[6]Cabe recordar que bajo este identificador se pueden recibir tanto notas como instrucciones sobre la partitura.

producción de la partitura y de cada nota, el de esta última a partir del tempo previamente indicado.

```
OS_Init();     // Init system variables

S.bEnable = 0;

// Create all tasks
OS_Task_Create(0, Task_INS);

OS_Task_Create(1, Task_CONDUCTOR); // this task must have lower priority

min = 0xFF;
max = 0x00;

OS_EI();       // Enable interrupts

OS_Run();      // Run OS kernel
```

Figura 5.14: Fragmento de la función principal del programa de los músicos en el que se inicia el OSA-RTOS y se crean las tareas a ejecutar.

5.1.5.3 Generación de sonido

Para la generación del sonido se cuenta con las funciones `InitSoundVariable` y `NoteWork`, ambas desarrolladas por Victor Timofeev. Ambas exigen como parámetro un puntero a una *struct* definida en el programa que contiene toda la información sobre el instrumento en cuestión (frecuencias de cada nota, forma de onda, nota actual y punto de inicio de repetición entre otras) y trabajan sobre la información que esta proporciona. La función `InitSoundVariable`exige además que se le pase como parámetro la cola de reproducción.

La función `InitSoundVariable` se utiliza para iniciar el sonido, reiniciando aquellos valores que hayan podido ser alterados por la reproducción de la nota previa.

La función `NoteWork`, en cambio, es la responsable de leer la nota o instrucción actual. Si es una instrucción, es aquí donde se traduce a una acción y se lleva a cabo. Si es una nota, en cambio, se traduce a la frecuencia correspondiente y se le aplica la forma de onda del instrumento correspondiente junto con el sintetizador, además de extraer su duración. Una vez realizado este procedimiento se obtiene la señal PWM que se emite por la salida correspondiente para ser reproducida por el altavoz.

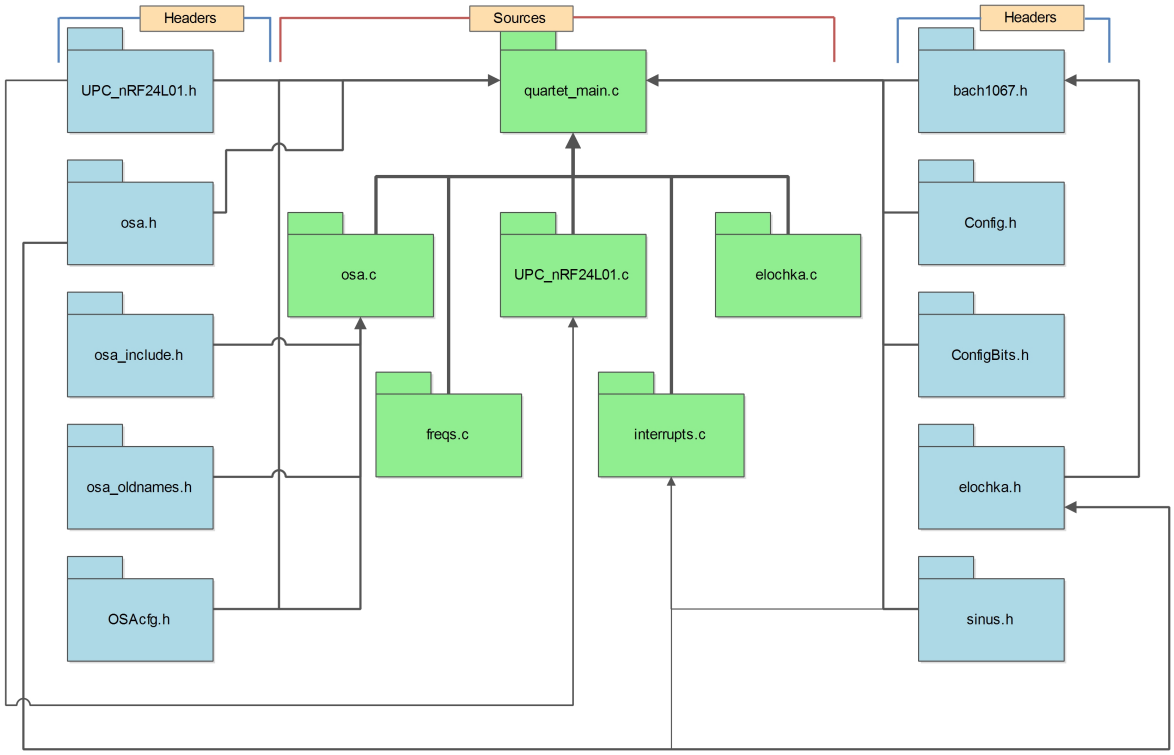

Figura 5.15: Estructura del programa de los músicos.

5.2 Desarrollo

El proceso seguido hasta la realización final de esta primera versión de la orquesta puede desglosarse en los siguientes hitos que han sido los que han ido marcando el avance del proyecto:

- Preparar las partituras y programar el director de la orquesta para que prepare los datos que contendrán los mensajes enviados por RF.
- Establecer una vía de comunicación por RF entre dos PIC18F4520 mediante el nRF24L01+.
- Establecer una vía de comunicación por RF entre la Pyboard y un PIC18F4520 mediante el nRF24L01+.
- Establecer una red de comunicación del tipo Maestro-Esclavos entre la Pyboard y dos PIC18F4520 mediante el nRF24L01+.
- Adaptar el código desarrollado por Joan Calvet en su TFG para el PIC18F4520 e incluir en él las rutinas de RF necesarias para la recepción de los paquetes enviados por el director.
- Desarrollar el código del director encargado de comunicarse con cada instrumento enviándole su partitura e instrucciones correspondientes.

- Incluir los dos músicos restantes y probar, ajustar y verificar el correcto funcionamiento de la orquesta.

6 Segunda versión de la orquesta

6.1 Diseño

La segunda versión de la orquesta se ha desarrollado sobre la primera versión antes presentada. Se ha partido del código producido para ella modificando y añadiendo aquello que fuera necesario con tal de implementar este nuevo paradigma de funcionamiento.

La motivación de esta segunda versión se basa en solventar las deficiencias de la primera versión y aumentar la versatilidad y flexibilidad de la orquesta a la hora de reproducir melodías. Las carencias que se quieren remediar son principalmente las limitaciones en la longitud de la partitura que supone almacenar toda la partitura en memoria y la poca flexibilidad en la comunicación entre microcontroladores que acarrea la transmisión de las partituras de forma secuencial músico tras músico.

De las tres fases expuestas en la primera versión de la orquesta (traducción de la partitura, comunicación vía RF y reproducción de la melodía) se han modificado las dos últimas. El formato de las partituras y su traducción a mensajes de un byte por parte del director no se ha modificado. En cambio, tanto la fase de comunicación como la de procesado y generación de sonido sí que se han visto modificadas.

La idea motriz tras esta segunda versión es que la interpretación de la melodía se lleve a cabo por parte de los músicos en tiempo real. Esto es, que sean capaces de reproducir la melodía a la vez que siguen recibiendo las notas e instrucciones que deben procesar a continuación.

Para esta segunda propuesta se ha definido un valor mínimo y un valor máximo del tamaño de la cola de reproducción, cola que, como en la versión anterior es una cola FIFO. Se ha preferido dejar un margen de notas entre la nota a tocar y la última nota recibida por motivos de seguridad. Este margen por eso, definido en la capacidad de la cola de reproducción, se ha dejado en manos del usuario. El tamaño mínimo es de una posición, aunque este tamaño no garantiza de ningún modo el correcto funcionamiento de la orquesta y, por lo tanto, no se recomienda su uso. El tamaño máximo, por contra, viene definido por la capacidad máxima de almacenamiento disponible en el microcontrolador. Para esta versión en concreto se ha definido un tamaño de cola de reproducción de un mínimo de 20 posiciones y de máximo 50. Estos valores no son, ni mucho menos, los únicos con los que se puede trabajar.

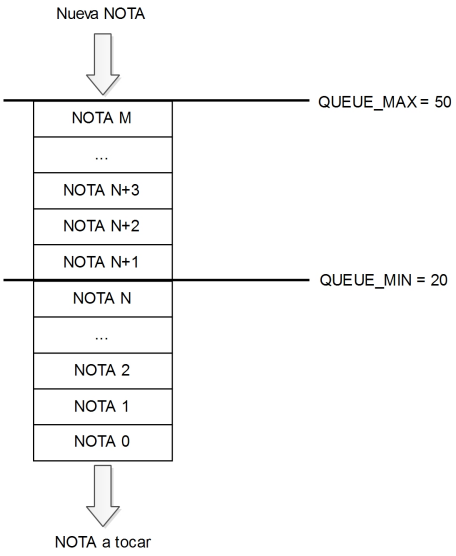

Figura 6.1: Estructura, funcionamiento y valores máximo y mínimo de posiciones de la cola FIFO definida para la segunda versión.

Definir la cola de reproducción de esta manera implica que por un lado el director ha de ser capaz de conocer el estado de la cola de cada de músico y, por otro lado, que los músicos han de ser capaces de comunicar al director el estado de sus colas. Para ello se han definido dos tipos nuevos de mensajes, uno para indicar que la cola no está llena y caben más notas, con lo que el director puede seguir enviando, y otro para indicar que la cola está completa, con lo que el director esperará un breve lapso de tiempo antes de enviar más notas a ese músico. Estos dos mensajes se han implementado mediante dos nuevos identificadores que son `FULL` y `EMPTY`, de valores 0x55 y 0xAA respectivamente.

Los músicos, que solo pueden comunicarse con el director después de que este haya iniciado la conversación, comunican el estado de sus colas en respuesta a los mensajes cuyo identificador es `NOTA_ID` o `STATUS_ID`. En el caso de que un músico le comunique al director que el estado de su cola es `FULL` el director dejará de enviarle notas y pasará a preguntarle por el estado hasta recibir como respuesta del músico que el estado de su cola es `EMPTY` para, acto seguido, retomar el envío de notas en el punto donde se había dejado.

Un aspecto crítico de esta versión, no como en la anterior donde una tarea sucedía a la otra mientras que en esta se realizan simultáneamente, es el tiempo necesitado para procesar la información y realizar cada tarea. Recordar aquí que el encargado de gestionar las tareas en los músicos es el sistema operativo OSA-RTOS. Las tareas han de realizarse a una velocidad suficientemente alta como para que no afecte a la velocidad de reproducción. Si esta última se viera alterada afectaría negativamente a la sincronización de la orquesta. Para evitarlo, las nuevas tareas implementadas se han dividido al máximo para reducir al máximo el tiempo que ocupan los recursos. También se han reducido al máximo los bucles de espera de las librerías utilizadas en el proyecto, por ejemplo el bucle de espera a la interrupción IRQ de la librería *UPC_nRF24L01*.

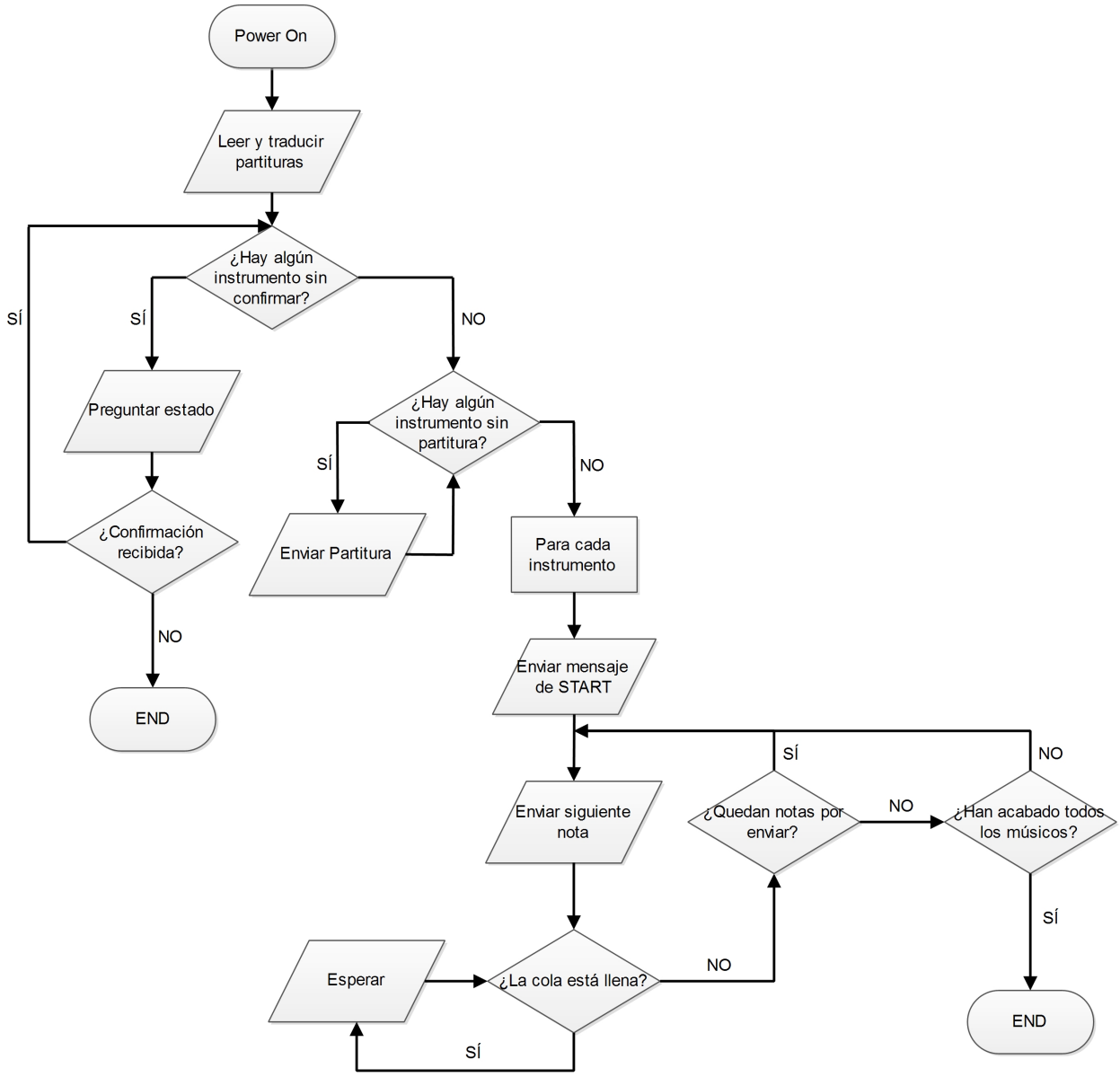

Figura 6.2: Flujo de ejecución del programa del director de la segunda versión de la orquesta.

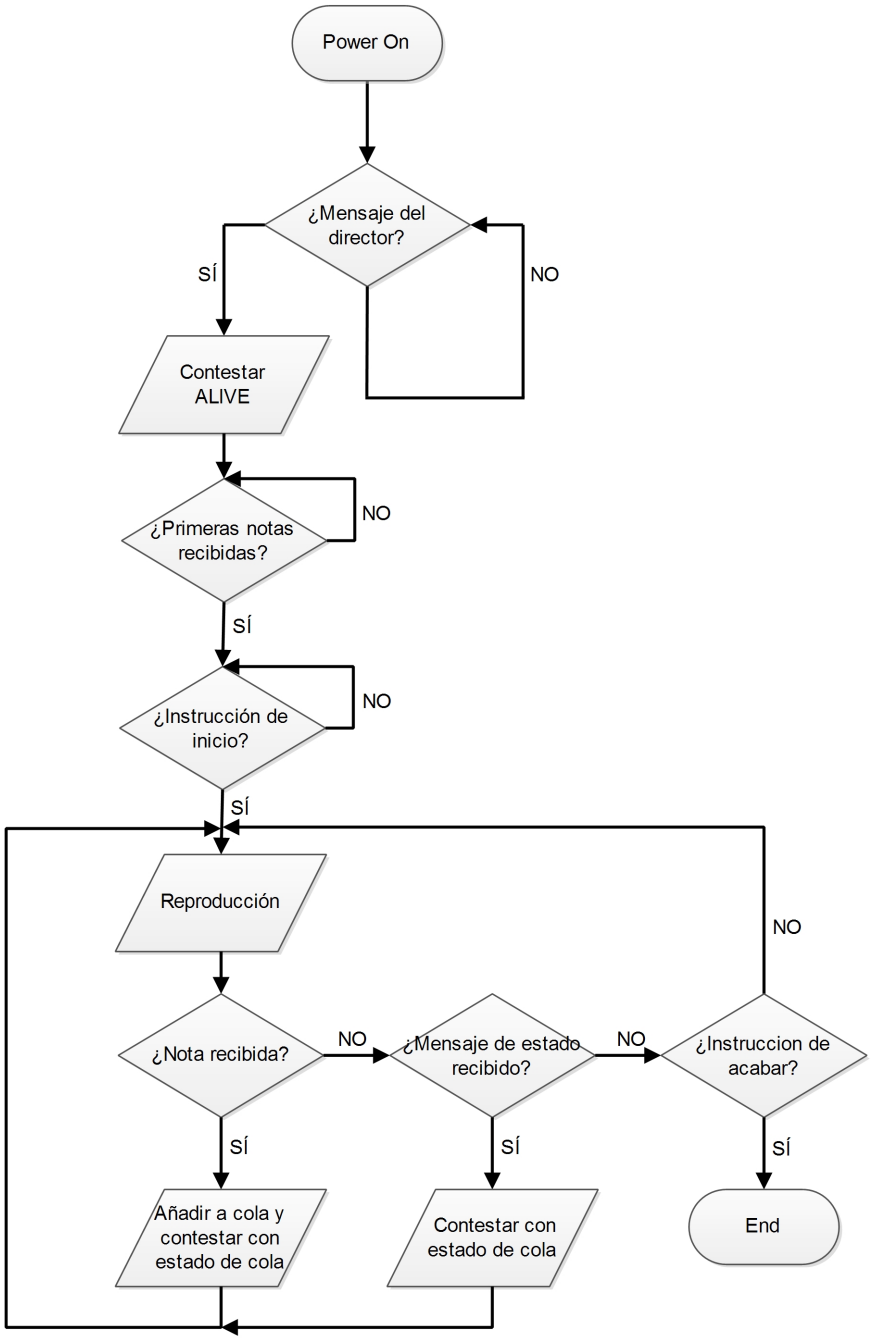

Figura 6.3: Flujo de ejecución del programa de los músicos de la segunda versión de la orquesta.

6.1.1 Código del director

Para esta segunda versión, del director solo se ha modificado el módulo *director.py* puesto que las tareas que realizan los otros dos módulos no varían en absoluto, solo cambia su aplicación, y de esta encarga el módulo *director.py*. Las principales modificaciones introducidas en el código son tres nuevos métodos, `send_data_to_one`, `send_next_note` y `ask_status` y cambios en la estructura de los ya existentes `main` y `send_notelist`, llamado ahora `send_notelist_n`. Además se han añadido dos nuevas propiedades a la clase `Conductor`, un *array* para guardar el estado de las colas de cada músico (`status`) y otro para guardar el puntero que identifica la siguiente nota a enviar a cada músico (`pointers_array`).

Los tres nuevos métodos desarrollados responden a las necesidades que han surgido a la hora de desarrollar esta versión.

El método `send_data_to_one` recibe cinco parámetros en el siguiente orden:

- `message_id` es el identificador del mensaje que se va a enviar. Se debe introducir en decimal.
- `payload` es el contenido propio del mensaje, de un byte de longitud. También se debe introducir en decimal.
- `instrument_tx` es la dirección del músico a quien va destinado el mensaje. Debe ser un *array* de bytes.
- `waiting_time` es el tiempo, en milisegundos, de espera para recibir una respuesta.
- `wait_for_response`, de carácter booleano, indica si se espera una respuesta o no por parte del músico.

Su función es enviar un mensaje que sólo tiene como destinatario a uno de los músicos y retornar la respuesta del músico, en el caso de que la haya y se espere.

Por otro lado, el método `ask_status`, desarrollado sobre el método anterior, recibe como único parámetro la dirección del músico del que se quiere conocer el estado. No retorna nada, pero actualiza el *array* de estados de los músicos al recibir la respuesta del músico.

El último de los nuevos métodos implementados, `send_next_note`, recibe también como único parámetro la dirección del músico a quien enviar la nota y, a partir de la partitura codificada del músico y el puntero que identifica la siguiente nota o instrucción a enviar se la envía. Una vez enviada actualiza tanto el puntero de nota de ese músico como su estado a partir de la respuesta recibida. Este método tampoco devuelve ningún valor.

La aparición de estos tres métodos se ve justificada por el planteamiento de esta versión. Mientras que en la primera versión la comunicación estaba determinada de antemano, y se sabía en todo momento cual era el siguiente mensaje a enviar y a quien ahora los mensajes enviados y su destinatario dependen del estado de las colas de reproducción y van variando en el tiempo sin un patrón determinado, con lo que hacen falta métodos más generales y flexibles que no estén encorsetados por culpa de una funcionalidad demasiado específica, si no que permitan comunicarse con un músico concreto en un instante cualquiera para transmitir el mensaje que haga falta en ese momento.

El método `send_notelist` de la versión anterior, encargado de enviar a cada músico su partitura, se ha modificado de manera que ahora permite especificar con un número entero que recibe como parámetro el número de notas e instrucciones que se quieren enviar al inicio a cada músico para poblar la cola de reproducción. Por eso se le ha cambiado el nombre a `send_notelist_n`.

En cuanto a la función `main`, encargada de gestionar la orquesta comunicándose con los músicos mediante RF, se ha modificado su código para cumplir con los requisitos establecidos en el diseño de esta nueva versión.

```
def main(self):

   while not self.check_all_alive():
      pass

   self.send_data_to_all(TEMPO_ID,self.tempo)

   for instrument_tx in self.direcciones_tx:
      self.send_notelist_n(instrument_tx,50)

   self.send_data_to_all(START_ID,0x00)

   while True:
      for direccion_tx in self.direcciones_tx:
         index=self.direcciones_tx.index(direccion_tx)
         if self.status_array[index] == EMPTY:
            self.send_next_note(direccion_tx)
         elif self.status_array[index] == FULL:
            self.ask_status(direccion_tx)
```

Figura 6.4: Función principal de la segunda versión de la orquesta.

Como se puede observar, el inicio es idéntico a la función principal de la versión previa, con la única diferencia que ahora, en vez de enviar la partitura entera se envían solo las primeras 50 notas a cada músico. Una vez hecho esto se procede a seguir enviando notas a cada músico mientras la respuesta recibida sea que la cola no está llena. Si alguno responde a una nota enviada que su cola está llena entonces el director pasa a preguntarle por el estado de la cola en vez de enviar notas. Cuando el músico contesta al director que la cola vuelve a tener posiciones libres el director retoma el envío de notas a ese músico hasta que se acabe la partitura.

6.1.2 Código de los músicos

Del código de los músicos se han modificado sólo los archivos *quartet_main.c*, donde se ha dotado al músico del comportamiento requerido para esta versión, y *UPC_nRF24L01.c*, donde lo único que se ha hecho ha sido reducir al mínimo los bucles de espera de la función de recepción de mensajes, `Receive_Data_RX_Mode_nRF24L01`, para agilizar el proceso y evitar que la orquesta pierda la sincronización por culpa de los tiempos de espera.

```
unsigned char nRF24L01_Status;
unsigned char Origin_Pipe;
unsigned char Checksum_Conclusion, IRQ_Error;
unsigned int i;
unsigned char j;

IRQ_Error=0;
CE=1;                                                          // Activate CE to
    leave Standby I mode and enter RX Mode (nRF24L01 searchs incoming signal)
Delay10TCYx(52);                                                 // TIMING
    CONDITION #2: Delay required between Stanby Modes and RX or TX Mode => 130 us
Delay10TCYx(4);                                                    // TIMING
    CONDITION #3: Minimum CE High => 10 us

i=0; j=0;
while(IRQ)                                                       // Wait until
    RX_DR is set High (Reception of Data), and then IRQ is set Low.
{                                                              // Sometimes the
    nRF24L01 fails to do this, so a maximum waiting time is needed
   i++;
   if(i==1)
   {
      i=0; j++;
   }
   if(j==Max_Waiting_ds)                                         // TIMING
       CONDITION #5: When we reach the Max_Waiting_ds defined by user, loop is then
       broken.
   {                                                           // If the device
       did not receive data before Max_Waiting_ds, IRQ ERROR is considered.
      IRQ_Error=1;                                             // The Result
          of the TX operation was: IRQ ERROR
      break;
   }
}
CE=0;                                                          // Deactivate CE
    to leave RX Mode (Low Current Consumtion)
Delay10TCYx(4);                                                    // TIMING
    CONDITION #4: Delay required between CE edge and CSN low => 4 us

nRF24L01_Status=Read_nRF24L01_Status();                                  //
    Update the nRF24L01 Status to get the Origin_Pipe
Origin_Pipe=(nRF24L01_Status & RX_P_NO);                               // Data
    Pipe Number is obtained from the STATUS Register
Checksum_Conclusion=Read_nRF24L01_RX_Payload(Enable_Checksum,TX_RX_Payload_Width,
    RX_Payload);  // After Reading the RX FIFO, all data is deleted from FIFO.
Write_nRF24L01_Status(RX_DR);                                    // Clear
    the RX_DR bit from the nRF24L01_Status Register

return (IRQ_Error*0b10000+Origin_Pipe+Checksum_Conclusion);
```

Figura 6.5: Cuerpo de la función de recepción de mensajes vía RF tras su modificación para la segunda versión de la orquesta.

```
void Task_comRF (void)
{

   for (;;){
   //OS_Yield();
   Start_RX_Mode_nRF24L01(0b11, 0b1000000, 0, 0b11, 1, 1, direccion_rx, 0x00, 0x00,
       0x00, 0x00, 0x00, 0, 2, 0, 0, 0, 0, 0);

   //OS_Cond_Wait(!(Receive_Data_RX_Mode_nRF24L01(0, 100, 2, mensaje_llegada)&
       0b00010000)); // Wait for RF message
   CE=1;
   OS_Cond_Wait(!IRQ);
   informe=Receive_Data_RX_Mode_nRF24L01(0, 100, 2, mensaje_llegada); //No Checksum
   Finish_nRF24L01_Operation();
   if(!(informe & 0b00010000))
   {
      if(mensaje_llegada[0]==NOTA_ID)
      {
            counter = counter + 1;
            cola_rx[i++] = mensaje_llegada[1];
            mensaje_rebotado[0]=EMPTY;
            mensaje_rebotado[1]=EMPTY;
            if (counter >= QUEUE_MAX)
            {
               mensaje_rebotado[0]=FULL;
               mensaje_rebotado[1]=FULL;
            }
            Start_TX_Mode_nRF24L01(0b11, 0b1000000, 0, 0b11, 1, 1, 0b0011, 15, 0, 2);
            Delay10OTCYx(100);
            Send_Data_TX_Mode_nRF24L01(0,0b11,direccion_tx,2,mensaje_rebotado);
            Finish_nRF24L01_Operation();
      }
      else if (mensaje_llegada[0] == STATUS_ID)
      {
         mensaje_rebotado[1]=FULL;
         mensaje_rebotado[0]=FULL;
         if (counter<=QUEUE_MIN)
         {
            mensaje_rebotado[0]=EMPTY;
            mensaje_rebotado[1]=EMPTY;
         }
         Start_TX_Mode_nRF24L01(0b11, 0b1000000, 0, 0b11, 1, 1, 0b0011, 15, 0, 2);
         Delay10OTCYx(100);
         Send_Data_TX_Mode_nRF24L01(0,0b11,direccion_tx,2,mensaje_rebotado);
         Finish_nRF24L01_Operation();
      }
   }
}
}
```

Figura 6.6: Tarea desarrollada para rastrear el aire en busca de mensajes en la segunda versión de la orquesta.

En cuanto al código del archivo principal, ahora se necesita una función que, gestionada por el sistema operativo, se encargue de ir rastreando el aire en busca de mensajes dirigidos a él, responder en concordancia, y actualizar el valor de sus variables si fuera necesario. Por lo tanto, en esta nueva versión se proponen tres tareas de las que se encargará OSA-RTOS. Las dos de la versión anterior y una nueva,`Task_comRF`, con la prioridad más baja de todas (2) y cuya misión es la acabada de explicar.

Nótese como el músico, independientemente del mensaje que reciba siempre contesta al director con el estado de su cola. Esta tarea se ha creado después en la función principal y se ha cedido el control sobre ella al OSA-RTOS. El resto de la función principal no se ha modificado. La gran dificultad de esta versión ha sido conseguir que los instrumentos no pierdan la sincronización, puesto que el tiempo de espera hasta recibir un mensaje es indeterminado, y si no son de varios ordenes de magnitud inferiores a los tiempos de duración de cada nota la orquesta va perdiendo la sincronización a medida que se avanza en la reproducción de la melodía. Para ello se han reducido a sus mínimos los tiempos máximos de espera de todas las funciones relacionadas con la recepción de datos por RF.

```
    // Create all tasks
    OS_Task_Create(0, Task_INS);

  OS_Task_Create(1, Task_CONDUCTOR);

  OS_Task_Create(2, Task_comRF);
```

Figura 6.7: Creación de las tres tareas que gestiona OSA-RTOS en la segunda versión de los músicos.

6.2 Desarrollo

A lo largo de la implementación de esta segunda versión han surgido varios problemas, alguno de los cuales ya se ha comentado, que se han tenido que ir solucionando. A continuación se exponen los principales problemas que han ido apareciendo y las soluciones que se les ha dado.

Un primer problema ha sido que los mensajes que enviaban los músicos comunicando el estado de sus colas al director no siempre llegaban se recibían de forma correcta al director, con lo que éste no era capaz de inferir es estado de cada músico. Después de probar diferentes alternativas sin encontrar la manera de solucionarlo se ha concluido que es probable que de vez en cuando, como explica Pablo Sanz en su PFC también, que la comunicación vía SPI entre el PIC18F4520 y el módulo nRF24L01 de vez en cuando falla y resulta en lecturas o escrituras erróneas. Por ello, la resolución final ha sido la de enviar en los dos bytes que forman el mensaje, identificador y datos, del músico al director el estado de la cola.

El otro problema principal que se ha encontrado ha sido el mencionado previamente de mantener la sincronización de la orquesta a lo largo de toda la reproducción de la pieza musical. Se ha encontrado al final que el problema era una mala gestión de los tiempos. La tarea implementada para recibir por RF en esta segunda versión copaba, a veces, los recursos del microcontrolador durante más tiempo del permisible. La solución ha sido reducir al máximo permisible los tiempos

de espera de todas las rutinas de RF utilizadas en la nueva tarea implementada.

6.3 Posibles mejoras

La principal mejora que se puede realizar en esta versión consiste en impelementar un sistema de cuenta atrás que sea el encargado de dar el señal de inicio a los músicos. De esta manera la sincronización inicial sería perfecta. Para ello el director debería iniciar la cuenta atrás mediante un mensaje de *broadcast* que marcaría, con su contenido, el tiempo restante para empezar la reproducción. Puesto que la tasa de errores cuando se envían este tipo de mensajes es muy elevada se debería ir enviando, cada cierto tiempo, otro mensaje mediante *broadcast* con el tiempo de cuenta atrás actualizado. Estos mensajes se envían con el objetivo de que al menos uno sea recibido por cada músico. Con que cada músico recibiera uno ya sabrían el tiempo restante para empezar a reproducir la melodía y la sincronización inicial entre músicos sería perfecta.

7 Presupuesto económico

SOFTWARE, COMPONENTES Y MATERIAL			
Ítem	Cantidad	Precio Unitario (€/u)	Coste Total (€)
Pack Open18F4520 Package B	4	76,51	306,04
Programador MPLAB ICD 2	1	180	180
PIC18F4520	4	8,63	34,52
Pyboard	1	37,36	37,36
Altavoces Sveon SON10	4	8,99	35,96
Resistencias 4,7KΩ	4	0,025	0,1
Resistencias 100Ω	4	0,025	0,1
Condensadores 100nF	4	0,07	0,28
Soldador de estaño	1	12,28	12,28
Bobina de estaño	1	1,73	1,73
Salidas de audio de 3,5mm	4	0,48	1,92
Regleta circuito impreso hembra con forma de codo de 40 pines	1	0,9	0,9
Programa MPLAB y compilador C18	1	0	0
Placa PCB para prototipaje de circuitos impresos	1	4,49	4,49
		TOTAL	615,68

HORAS DE TRABAJO			
Concepto	Horas	Coste Unitario (€/h)	Coste Total (€)
Estudio previo	75	12	900
Desarrollo	295	12	3540
Validación	40	12	480
		TOTAL	4920

CONSUMO DE RECURSOS ENERGÉTICOS			
Causa del consumo	Horas	Coste Unitario (€/kWh)	Coste Total (€)
Electricidad de los componentes electrónicos	335	0,14	46,9
Electricidad del lugar de trabajo	425	0,14	59,5
		TOTAL	106,4

COSTE PROYECTO	
Gasto	Coste (€)
Software, componentes y material	615,68
Horas de trabajo	4920
	106,4
TOTAL	**5642,08**

Para valorar y calcular el coste total del proyecto se han tenido en cuenta tres ámbitos que se ha considerado que son los más influyentes del proyecto en el aspecto económico. Estos son el material, los componentes electrónicos y el software, las horas invertidas y los recursos energéticos consumidos. Puede observarse que el coste total estimado es de unos 5650 € repartidos sobretodo entre el coste del material y componentes y las horas invertidas. Dentro de el coste de los materiales y componentes hay que destacar que la gran mayoría de ellos es reutilizable, con lo que su coste se puede ir amortizando con el paso del tiempo y el repetido uso que se puede hacer de ellos.

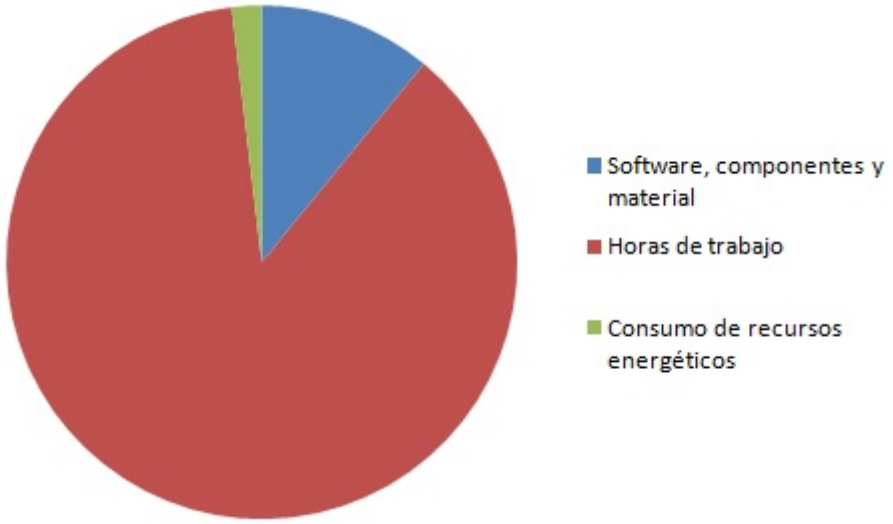

Figura 7.1: Distribución porcentual de los costes del proyecto.

8 Impacto medioambiental

El impacto medioambiental del proyecto debe evaluarse desde dos puntos de vista que se complementan, aportando cada uno datos que ayudan a obtener una visión global de este. Por un lado está el ciclo de vida de los materiales y componentes empleados en el proyecto, y por otro la alteración del medio que este puede provocar, incluyendo en este segundo ámbito a las personas y al medio físico alrededor del proyecto durante su utilización.

Respecto al ciclo de vida de los componentes y materiales utilizados, debe partirse de la premisa de alargar al máximo la vida útil de todos y cada uno de ellos y de utilizar sólo aquellos que sean imprescindibles. Dicho esto, una vez alcanzado el final de su vida útil, para la correcta gestión del reciclado y de los desechos se deberá llevar los componentes a algún centro de gestión de residuos de aparatos eléctricos y electrónicos (RAEE). Allí se separan, principalmente con procesos mecánicos, los elementos contaminantes de los no contaminantes. Los primeros se gestionan de forma apropiada mientras que los segundos se acumulan y transforman en materias primas aprovechables. En caso contrario el contenido en materiales contaminantes de estos componentes (PVC, PCB o TBBA entre otros) será altamente perjudicial para el medio ambiente. Aproximadamente el 70% de cada dispositivo se puede volver a aprovechar.

Se concluye por tanto que el impacto medioambiental de cada dispositivo a lo largo de su ciclo de vida depende en gran medida de la manera en que se gestiona tanto su uso como su trato como residuo. En la medida en que se alargue al máximo su vida útil y reciba el trato adecuado como residuo electrónico en una planta capacitada para gestionar de forma integral los RAEE se puede reducir en gran medida su impacto medioambiental.

En cuanto al impacto medioambiental de los dispositivos utilizados en este proyecto, debe considerarse como causa principal las ondas de RF emitidas en la comunicación entre los distintos microcontroladores. El efecto que esta pueda tener sobre su entorno esta ligado principalmente a cuatro parámetros:

- El tiempo de exposición a la radiación.
- El alcance y rango de las ondas de Radio Frecuencia.
- La potencia generada por la antena del módulo nRF24L01+.
- Frecuencia electromagnética de la onda con que se trabaja.

El tiempo de funcionamiento de la orquesta varía en función de la versión utilizada y de la longitud de la partitura. Aún y así, la variación del tiempo de exposición a la radiación entre

versiones es mínimo, pues se puede garantizar que el tiempo durante el que el nRF24L01+ está emitiendo radiación es de, como máximo, 102 mS por mensaje.

$$\frac{81\frac{bits}{mensaje}}{10^6\frac{bits}{S}} + 14\ intentos \cdot \frac{4750\frac{\mu S}{intento}}{10^{-6}\frac{S}{\mu S}} = 68\ mS$$

$$68\ mS \cdot \frac{15}{10}\ coeficiente\ de\ seguridad = 102\ mS$$

Suponiendo un número razonable de mensajes, 500 por instrumento, resultaría en tiempo de exposición, en el peor de los casos[1], de 3 minutos y 24 segundos.

$$500\frac{mensajes}{instrumento} \cdot 4\ instrumentos \cdot 102\frac{mS}{mensaje} = 204000\ mS = 3\ minutos\ y\ 24\ segundos$$

Este dato, que de entrada puede parecer elevado, debe verse a la luz del resto de factores para conocer su gravedad real.

La frecuencia de trabajo del módulo de Radio Frecuencia oscila entre los 2.4 y 2.5 GHz, situada entre los 3 MHz y los 3 GHz que componen toda la banda del espectro electromagnético. Es cierto que a potencias elevadas las ondas con frecuencias dentro de este espectro pueden causar daños a un individuo tal y como alerta la OMS. Por eso se procede ahora a calcular la potencia generada por la antena del módulo nRF24L01+ con el fin de enviar un mensaje por Radio Frecuencia.

La potencia generada en el campo electromagnético de salida de la antena del módulo nRF24L01+ trabajando a 0 dBm según la configuración definida anteriormente corresponde a una potencia de 1 mW.

$$P_{dBm} = 10 \cdot log_{10} \cdot P_{mW} \quad \Rightarrow \quad P_{mW} = 1\ mW$$

Dentro del rango de frecuencias de entre 1 MHz y 10 GHz, según la Organización Mundial de la Salud, es necesario un SAR de al menos 4 W/Kg para afectar a la salud individual de manera grave. En una persona de 75 Kg este límite se situaría entonces en 300 W y 1 mW representa un 0,0000033% de este valor límite, con lo que se puede considerar nula su influencia sobre la salud humana. Además este valor se reduce al alejarse la persona del foco de emisión y, con un alance máximo aproximado de 30 metros[2] el radio de acción del módulo es muy reducido.

En vista de los datos expuestos se puede garantizar con toda seguridad, a pesar del tiempo de exposición previamente calculado, que la transmisión por radiofrecuencia mediante el módulo nRF24L01+ no presenta ningún tipo de peligro para ningún ser humano.

[1] Nótese que el tiempo de exposición en el caso ideal es de más de 15 veces inferior.

[2] Pablo Sanz, en su PFC recomienda no alejar los módulos más de 10 metros pues entonces la señal ya es muy débil y hay muchos errores de transmisión.

9 Conclusiones

Una vez finalizado y validado el correcto funcionamiento de la orquesta mediante la reproducción de las piezas musicales propuestas en este proyecto se puede concluir que se han superado de forma satisfactoria los objetivos planteados al inicio del proyecto.

Se han tenido serias dificultades para establecer con éxito una vía de comunicación mediante Radio Frecuencia entre los microcontroladores presentes en el proyecto. Los problemas se debían principalmente a una incorrecta configuración de los parámetros de la transmisión de información a través del bus SPI y del módulo nRF24L01+. Su resolución no ha sido fácil puesto que no se contaba con todo el instrumental necesario para determinar las causas de los problemas que han ido surgiendo. Se ha echado sobretodo de menos un *sniffer* de RF para determinar que pasa en el aire cuando hay fallos de transmisión. La metodología que se ha seguido para resolver los problemas y que ha resultado más efectiva, aunque lenta, ha sido la aproximación de "divide y vencerás". Una primera conclusión del proyecto aunque sin relación directa con él es que, en los trabajos con dispositivos electrónicos, es tan importante elegir los componentes adecuados como disponer del instrumental necesario para poder determinar de manera fiable en cualquier momento qué está pasando.

Finalmente, se ha conseguido establecer una comunicación fiable y eficaz mediante Radio Frecuencia entre microcontroladores a través del módulo nRF24L012+ en la que no se dan pérdidas de información ni referencias cruzadas gracias a una correcta configuración del módulo y un poco de creatividad. Es cierto que esta configuración ha disminuido notablemente la velocidad de transmisión respecto a la velocidad máxima facilitada por el fabricante, pero ha sido en favor de la fiabilidad. Se ha considerado que esta era más importante que la velocidad. Además,la velocidad finalmente utilizada es más que suficiente para la aplicación concreta aquí desarrollada.

A partir de la experiencia con el módulo nRF24L01+ se puede afirmar que, a pesar de las dificultades que pueda presentar su utilización al principio, es muy cómodo trabajar con él. Exceptuando la parte en la que se explican las *pipes* y su uso, en el manual está todo muy detallado y bien explicado y además pueden encontrarse en Internet un gran número de proyectos realizados con él que facilitan la compresión de su funcionamiento. Es un módulo ideal para iniciarse en la comunicación por RF y que presenta un muy buen rendimiento en las tareas para las que ha sido diseñado. El problema que se ha encontrado es que es poco flexible a la hora de permitir al usuario definir y diseñar su propio sistema y estructura de comunicación.

En cuanto a la orquesta, se ha desarrollado tanto un sistema de comunicación y reproducción en tiempo real como en diferido que cumple con los requisitos necesarios para garantizar el buen funcionamiento de la misma. Haber realizado las dos facilita que la comparación entre los dos

sistemas sea objetivo y las conclusiones extraídas dignas de crédito. El hecho de trabajar en diferido presenta la limitación de la capacidad de la memoria de los microcontroladores mientras que el sistema en tiempo real tiene una capacidad ilimitada debido al procesado inmediato de los mensajes entrantes. El hecho de que esta versión no presente esta restricción de memoria y los músicos funcionen de forma inalámbrica abre un gran abanico de posibilidades no solo en cuanto a posibles aplicaciones si no que también en cuanto a la ubicación y facilidad de instalación de los músicos. El problema que tiene es que la probabilidad de fallos en la comunicación es más elevada que en la primera propuesta de la orquesta.

La conclusión una vez comparados los dos sistemas es que para aplicaciones donde la respuesta no es inmediata y no es importante el flujo constante de información para garantizar un funcionamiento correcto no vale la pena trabajar con un sistema de comunicación y procesado en tiempo real. En el caso de que si se requiera trabajar con una comunicación y procesado en tiempo real se recomienda primero buscar una alternativa híbrida o, como mínimo, buscar que puntos se pueden aprovechar de la comunicación en y procesado en diferido puesto que facilitará mucho la implementación de la aplicación.

Bibliografía

[1] Brennen Ball. Everything You Need to Know about the nRF24L01 and the MiRF-v2, 2007. Tutoriales 0,1,2 y 3.

[2] Arístides Barea. Análisis de las prestaciones de la placa Micropython v.1.0. Trabajo de Fin de Grado, Escola Tècnica Superior d'Engeniería Industrial de Barcelona, Septiembre 2015.

[3] Damien George. *Micro Python Documentation Release 1.3.6*, Noviembre 2014.

[4] Damien George. nRF24L01 driver for Micro Python. En línea, Septiembre 2015. https://github.com/micropython/micropython/tree/master/drivers/nrf24l01.

[5] Nordic Semiconductor. *nRF24L01+ Single Chip 2.4GHz Transceiver Preliminary Product Specification v1.0*, Marzo 2008.

[6] Pablo Sanz. Comunicación por RF entre microcontroladores PIC18 mediante el módulo NRF24L01. Proyecto de Final de Carrera, Escola Tècnica Superior d'Engeniería Industrial de Barcelona, Junio 2014.

[7] Victor Timofeev. Osa documentation. En línea, Octubre 2010. http://www.pic24.ru/doku.php/en/osa/ref/intro.